Facultade de Informática
UNIVERSIDADE DA CORUÑA

TRABALLO FIN DE GRAO
GRAO EN ENXEÑARÍA INFORMÁTICA
MENCIÓN EN ENXEÑARÍA DE COMPUTADORES

Aplicación móvil colaborativa para localizar y etiquetar baños de acceso público

Estudante: Matías Cubilla Currás
Dirección: Carlos Vázquez Regueiro

A Coruña, febreiro de 2020.

”¿Que si lo entiendo? ¡Claro que lo entiendo! Hasta un niño de cuatro años podría entenderlo.
¡Traedme a un niño de cuatro años!”
Duck Soup(1933)

Agradecimientos

A mis padres, aunque no se lo diga lo suficiente.

A mi hermano, porque sin él sería hijo único.

A mis amigos, por el apoyo incondicional.

A mis compañeros, porque el viaje puede hacerse largo.

A mi tutor, Carlos, por su implicación y dedicación, no solo como tutor, sino también como profesor.

Y a la educación pública, porque sin ella yo no habría llegado a realizar estudios superiores.

Resumen

Este proyecto tiene como objetivo crear un sistema con la información de los aseos públicos que se encuentran en las cercanías del usuario. La aplicación es colaborativa, de forma que un usuario registrado puede crear nuevos registros, editarlos, hacer comentarios y poner valoraciones. De este modo, los usuarios tienen un acceso actualizado y veraz a la información relevante sobre los aseos públicos. El diseño ha sido modular y fácilmente extensible a otros recursos geolocalizados, como parques o establecimientos deportivos.

El sistema consta de dos elementos, un servidor y un cliente móvil. El servidor está implementado en Firebase y gestiona el acceso. Al mismo tiempo almacena la información relevante para la interacción y el correcto funcionamiento del sistema. Además gestiona las geolocalizaciones de los espacios y los datos relacionados a cada uno, como comentarios y valoraciones. El cliente móvil es una aplicación Android con una interfaz fácil de usar, intuitiva, y al mismo tiempo atractiva. Con ella los usuarios pueden acceder y, en su caso, editar la información almacenada en el sistema.

Abstract

The project's objective is to create a system of public toilet's information that can be found around the user's position. The application is collaborative, so any authenticated user can create new registers, edit them, or even make comments and put ratings. In this way, the user always have a truthful and updated access to the information about the public toilets. The design has been modular and easily extendable to any other geolocatable resource, like parks or sports facilities. The system is composed by two elements, a server and a mobile client. The server is implemented in Firebase and manages the access. At the same time It saves the relevant information for the interaction and the proper functioning. Also, It manages the space's location and the relevant data about each one like comments or ratings. The mobile app is the Android's application with an easy to use interface, intuitive, and at the same time attractive. With the app the user can access and edit the stored data in the system.

Palabras clave:

- Aplicación Android
- Aplicación colaborativa
- Recursos geolocalizados
- Firebase

Keywords:

- Android application
- Collaborative application
- Geolocatable resources
- Firebase

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

En este primer capítulo de la memoria se hará una breve introducción al proyecto. Se comenzará mostrando las motivaciones que han guiado el desarrollo de este proyecto. A continuación, se enumeran los principales objetivos que se pretenden implementar y se describirán brevemente las propuestas. Cerramos el capítulo con una breve explicación de la estructuracion de toda la memoria.

1.1 Motivación

Hoy en día es innegable considerar que gran parte de nuestro tiempo, y más en ciudades grandes o en vacaciones, lo empleamos en traslados de un punto a otro. Por esto, en medio de estos trayectos, muchas veces nos es necesario utilizar o al menos saber que existe un aseo público en condiciones al que poder acceder en las cercanías. Por supuesto este tipo de necesidad se ve incrementada considerablemente cuando se está enfermo, o incluso se tienen unas necesidades de accesibilidad más específicas, como puede ser que posea una rampa, que tenga un tamaño suficiente, o que tenga una barra de apoyo, o incluso, un cambiador de pañales.

Aunque ya existen aplicaciones para encontrar baños en las cercanías de nuestra posición (*'Where is Public Toilet'*, *'Toilet finder'*, *'Buscar baño'*, etc), de las cuáles hablaremos más en detalle en el capítulo 2, en general carecen de tres principales características:

- No existe ningún tipo de realimentación ni interacción con el usuario.
- Sólo se centran en un apartado funcional muy concreto, dejando de lado información importante para el usuario.
- No son aplicaciones intuitivas.

Todo esto sumado, abre la puerta a la idea de crear una aplicación que preste un servicio a una necesidad real, que sea intuitiva, fácil de usar, y que proporcione las herramientas para

que el usuario se sienta parte de la evolución y calidad de la información proporcionada por la misma.

Se propondrá un diseño modular de la aplicación de modo que en un futuro próximo sea muy fácil añadir nuevas funcionalidades o generalizar el sistema para orientarlo a otro tipo de servicios públicos o privados: parques infantiles, parques para mascotas, jardines de infancia, etc.

1.2 Objetivos

Por tanto, este proyecto tiene como objetivos primordiales:

- Desarrollar una aplicación móvil siguiendo un arquitectura que facilite el trabajo con tecnologías Android, así como también la escalabilidad y versatilidad del código.
- Arquitectura general pero aplicada al caso de gestionar información sobre baños: geoposición, características funcionales (rampas, barra, acceso, etc), propiedades (tamaño, papel, secadores, jabón, etc.) y otros atributos (limpieza, calidad), etc.
- La aplicación tendrá un modo colaborativo que permita a los usuarios actualizar la información sobre cada baño, hacer valoraciones y poner comentarios.
- Aplicación con una interfaz intuitiva y fácil de usar.

1.3 Propuesta

Para alcanzar los objetivos marcados en el apartado anterior, el proyecto se cemtrará en desarrollar los siguientes elementos:

- Una infraestructura online centralizada que incluye un servidor con una base de datos, para el almacenaje, compartición de datos y despliegue a la aplicación.
- Una aplicación Android que ofrezca una interfaz intuitiva a la vez que llamativa y con las funcionalidades necesarias.
- Un sistema de localización basado en el sensor GPS de cada dispositivo móvil.

1.4 Estructura de la memoria

En los siguientes apartados se detallarán los distintos aspectos que han conformado el desarrollo del proyecto. A continuación una breve descripción de cada uno de ellos.

- Estado del arte: Este capítulo analiza el estado actual de los aspectos más importantes de nuestra aplicación en la actualidad.
- Fundamentos tecnológicos: Este capítulo se centra en explicar las tecnologías utiñizadas a lo largo del desarrollo del pryoecto.
- Análisis y requisitos: Este capítulo tratará en análisis del proyecto desde el punto de vista de los requisitos, actores y casos de uso.
- Metodología de desarrollo: En este capítulo se explica y se razona la elección de la metodlogía empleada, como también su adaptación al proyecto.
- Arquitectura y modelo de datos: En este capítulo se explica el diseño de la arquitectura de sistema y de aplicación a implementar, como también el diseño del modelo de datos.
- Implementación: En este capítulo trataremos los detalles más destacables o particulares de la implementación de la arquitectura y el modelo de datos diseñado.
- Planificación y costes: En este capítulo se tratará la planificación inicial del proyecto, su seguimiento, y los costes finales que han supuesto.
- Pruebas: Capítulo en el que se detallan las pruebas realizadas.
- Conclusiones y trabajo futuro: En esta capítulo se cierra la memoria con las conclusiones y las posibilidades de trabajo futuro del proyecto.

Capítulo 2

Estado del arte

En este apartado realizaremos un breve estudio del estado actual del mercado de aplicaciones móviles relacionadas con la localización de distintos servicios permitiendo la actualización colaborativa de la información de cada uno de ellos, y con la localización de servicios públicos.

2.1 Localización de servicios con actualización colaborativa de la información

Dentro del mercado de aplicaciones nos encontramos con una gran cantidad de ellas enfocadas a aportar información a partir de una localización.

2.1.1 Características generales

Dentro de esta categoría, lo principal que encontramos son aplicaciones enfocadas a la hostelería, donde la más conocida es *TripAdvisor* y también podríamos incluir a *GoogleMaps* ya que además de localizar servicios, da la posibilidad de agregar valoraciones y comentarios, aunque ésta última ya no se enfoca solo a la hostelería. De estas aplicaciones podemos sacar una serie de ideas principales:

- Los usuarios tienen la posiblidad de puntuar de forma general a los servicios, y en el caso de TripAdvisor además, se pueden puntuar una serie de categorías como *Comida*, *Servicio*, *Calidad/Precio* y *Atmósfera*.
- Permiten agregar comentarios a mayores de la valoración.
- Permiten dejar preguntas para la persona o personas que gestionan el servicio.
- Ofrecen información a mayores que puede resultar útil, como número de teléfono, página web y en el caso de Google Maps las posiblidades de transporte público para llegar

al sitio.

- Permiten subir imágenes sobre los sitios.

2.1.2 Principales aplicaciones

Además de las aplicaciones ya mencionadas, existen otras muchas que se se basan en un comportamiento muy parecido, y como ya ha sido comentado. Algunas de las más resaltables son:

- **TripAdvisor** Como ya hemos comentado, es una aplicación que se centra en todo tipo de servicios que giran en torno a viajes. Además de las características ya comentadas, permite la posibilidad de contratar o reservar servicios a través de la app [1].
- **GoogleMaps** También comentada antes, no se centra sólo en hostelería y deja valorar desde servicios, empresas o sitios turísticos a partir de su localización [2].
- **FourSquare** Es una aplicación con un funcionamiento muy similiar a TripAdvisor, pero centrado puramente en hostelería [3].
- **Waze** Aplicación para facilitar el los viajes en coche permitiendo a los usuarios poner información a partir de localizaciones de distintas circunstancias que se puedan dar; carretera cerrada, mucho tráfico, accidente, etc... [4].

2.2 Localización de aseos públicos

Dentro de esta categoría, aunque no nos vayamos a encontrar una gran cantidad de aplicaciones, sí que encontramos algunas, de las cuáles podríamos destacar las siguientes

2.2.1 *Where is Public Toilet*

Where is Public Toilet es una aplicación que ofrece la posibildad de buscar baños en las cercanías, crear localizaciones donde se encuentren baños y también aporta cierta información útil sobre cada sitio [5]. Podemos ver un ejemplo de alguna de sus pantallas en la figura 2.1.

Es un buen ejemplo de lo que se puede buscar, pero tiene una serie de problemas:

- Solo se tiene en cuenta la última información añadida a la localización.
- Es necesario descargar la base de datos sobre una zona para poder usarla.
- No permite añadir comentarios ni valoraciones
- No es intuitiva

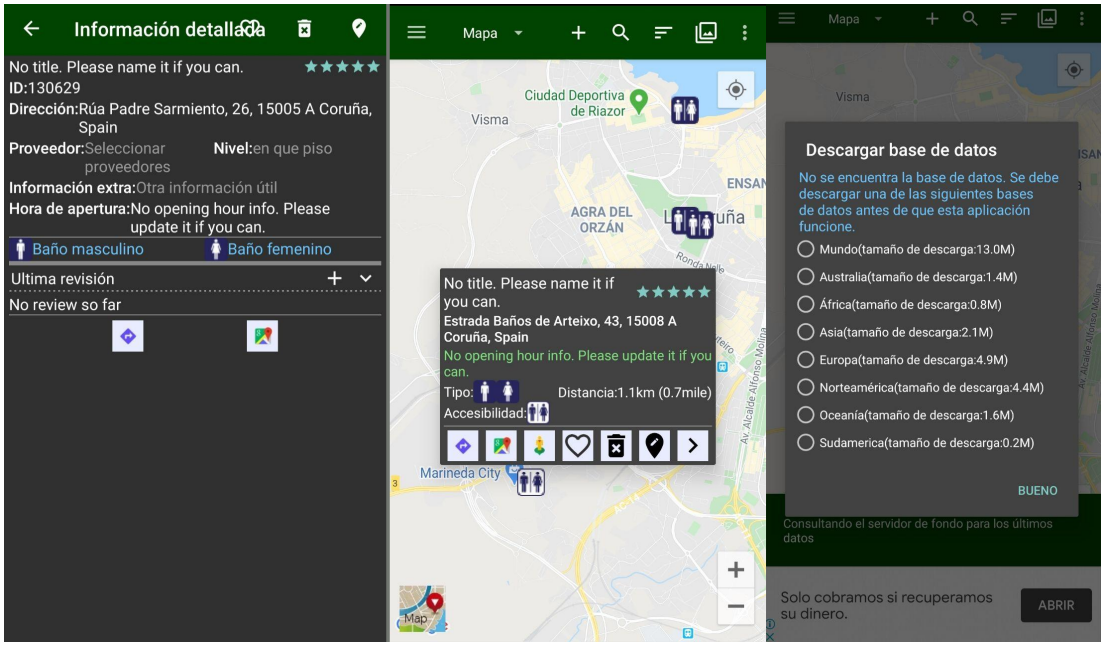


Figura 2.1: Capturas de la aplicación *Where is public toilet*

2.2.2 *Aseos cerca*

Aseos cerca se centra en localizar los baños que se encuentran cerca de la localización actual [6].

Es un ejemplo un tanto más pobre que el anterior, ya que aunque soluciona problemas como el de tener que descargar una base de datos de la zona, tiene otros problemas.

- Solo indica la localización de cada baño, ninguna información a mayores.
- La única forma de añadir localizaciones es enviando un correo informando de que falta en una localización.
- No es intuitiva

De esta aplicación podemos ver una serie de ejemplos de sus pantallas en la figua 2.2.

2.2.3 Otras aplicaciones

Existen más ejemplos de aplicaciones para localizar baños, todas con funcionalidades muy parecidas, pero todas con las mismas problemáticas mencionadas anteriormente, principalmente hablamos de aplicaciones poco intuitivas a la hora de usar. A continuación enumeramos algunas de ellas sin entrar en detalles: *WC Rome*, *Toilet Finder*, *Toilets WC*, *ToiFi* o *Buscar Baño*

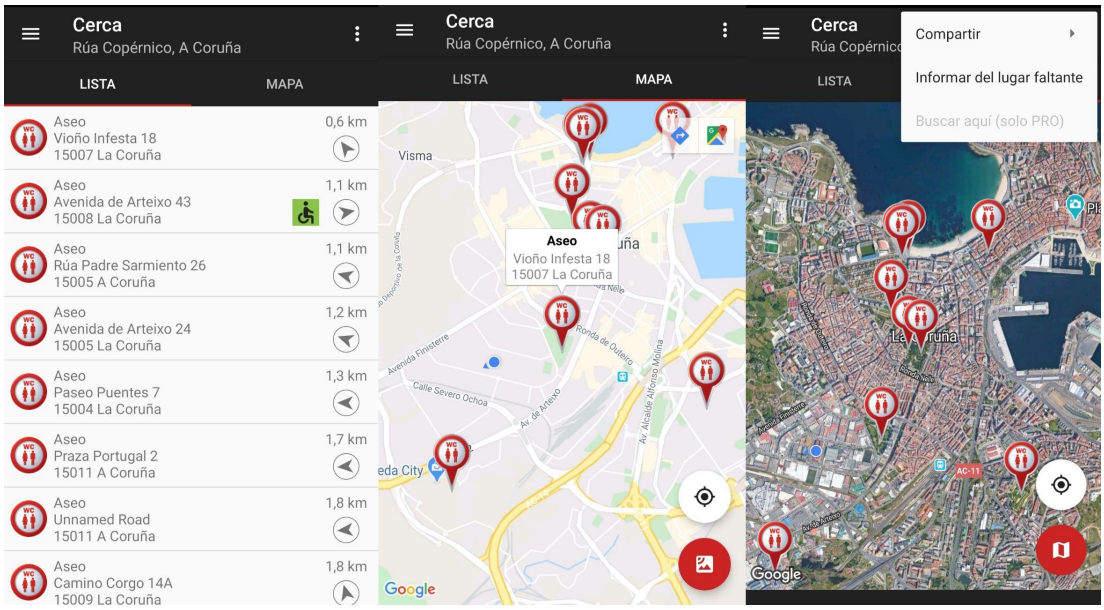


Figura 2.2: Capturas de la aplicación *Aseos cerca*

Capítulo 3

Fundamentos tecnológicos

A lo largo de este capítulo se introducirán las tecnologías utilizadas para llevar a cabo el proyecto. Comenzaremos por explicar el entorno Android, el entorno para el que está orientada nuestra aplicación. A continuación se describirá la infraestructura Firebase, clave para nuestro proyecto, ya que incorpora herramientas para autenticar usuarios, y también para almacenar y compartir información, incluso en tiempo real. También se comentarán las principales librerías Android y herramientas de edición, programación, documentación y gestión empleadas durante el desarrollo del proyecto.

3.1 Android

El sistema operativo Android es un sistema Linux multiusuario en el que cada aplicación es un usuario diferente. A continuación explicaremos su arquitectura a nivel de sistema operativo y también la estructura que tienen sus aplicaciones.

3.1.1 Arquitectura

Android es una pila de software de código abierto basado en Linux creada para una variedad amplia de dispositivos y factores de forma. En el siguiente diagrama, se muestran los componentes principales de la plataforma Android [7].

La base de la plataforma Android es el kernel de Linux, siguiéndole por encima la capa de abstracción de Hardware(HAL). La capa de abstracción de hardware (HAL) brinda interfaces estándares que exponen las capacidades de hardware del dispositivo al marco de trabajo de la API de Java de nivel más alto. La HAL consiste en varios módulos de biblioteca y cada uno de estos implementa una interfaz para un tipo específico de componente de hardware.

El siguiente nivel está dividido en dos partes; por una parte el Android Runtime, que para los dispositivos con Android, cada app ejecuta sus propios procesos con sus propias instancias del tiempo de ejecución de Android (ART). El ART está escrito para ejecutar varias máquinas

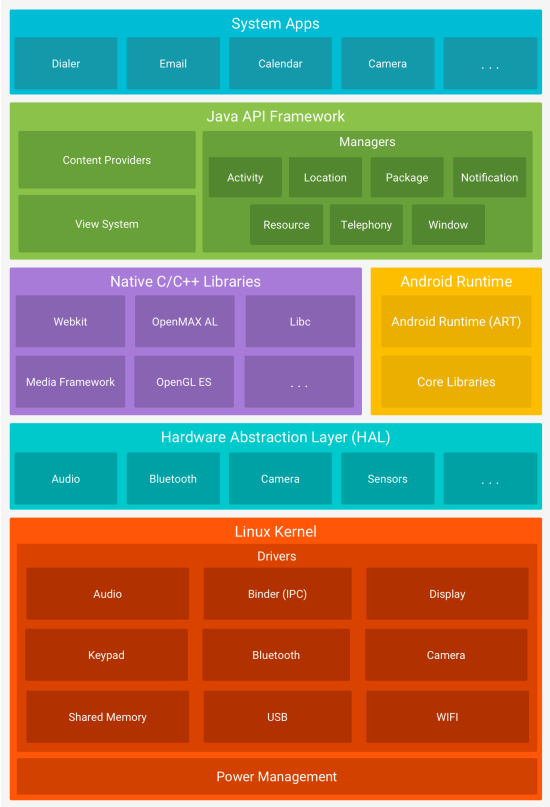


Figura 3.1: Arquitectura del sistema operativo Android

virtuales en dispositivos de memoria baja ejecutando archivos DEX, un formato de código de bytes diseñado especialmente para Android y optimizado para ocupar un espacio de memoria mínimo; y por otro lado las Bibliotecas C/C++ nativas, cuya necesidad está basada en que muchos componentes y servicios centrales del sistema Android, como el ART y la HAL, se basan en código nativo que requiere.

Todo el conjunto de funciones del SO Android está disponible mediante API escritas en el lenguaje Java. Estas API son los cimientos que necesitas para crear apps de Android simplificando la reutilización de componentes del sistema y servicios centrales y modulares, y de esta parte se encarga la Java API Framework.

Y por último tenemos las aplicaciones de sistema que son todo el conjunto de apps centrales para correo electrónico, mensajería SMS, calendarios, navegación en Internet y contactos, entre otros elementos.

3.1.2 Componentes

Las aplicaciones son el centro de Android a nivel de usuario. Solo una aplicación puede estar en primer plano al mismo tiempo y se ejecuta, por defecto, en un solo hilo de ejecución.

Cada aplicación tiene al menos uno, o varios, de los siguientes componentes [8]:

- Actividad: Una actividad es el punto de entrada de interacción con el usuario. Representa una pantalla individual con una interfaz de usuario. Por ejemplo, una aplicación de correo electrónico tiene una actividad que muestra una lista de los correos electrónicos nuevos, otra actividad para redactar el correo electrónico y otra actividad para leerlo. Si bien las actividades funcionan juntas para ofrecer una experiencia de usuario uniforme en la aplicación de correo electrónico, cada una es independiente de las demás.

- Servicio: Un servicio es un punto de entrada general que permite mantener la ejecución de una aplicación en segundo plano por diversos motivos. Es un componente que se ejecuta en segundo plano para realizar operaciones de ejecución prolongada o para realizar tareas de procesos remotos. Un servicio no proporciona una interfaz de usuario.

- Receptores de emisiones: Un receptor de emisión es un componente que posibilita que el sistema entregue eventos a la aplicación fuera de un flujo de usuarios habitual, lo que permite que la aplicación responda a los anuncios de emisión de todo el sistema. Dado que los receptores de emisión son otro punto bien definido de entrada a la aplicación, el sistema puede entregar emisiones incluso a las aplicaciones que no estén en ejecución.

- Proveedores de contenido: Un proveedor de contenido administra un conjunto compartido de datos de la aplicación que puedes almacenar en el sistema de archivos, en una base de datos SQLite, en la Web o en cualquier otra ubicación de almacenamiento persistente a la que tenga acceso tu aplicación. A través del proveedor de contenido, otras aplicaciones pueden consultar o modificar los datos si el proveedor de contenido lo permite.

3.2 Firebase

Firebase [9] es una infraestructura online que ofrece una serie de herramientas para el desarrollo de aplicaciones móviles y web. Entre esas herramientas destacan dos en cuanto a relevancia para nuestro proyecto: un sistema de autenticación y una base de datos en la nube.

Aunque existen otras posibilidades como *MongoDB* [10] o *PostgreSQL* [11], utilizar *Firebase*, además de implicar un despliegue y una conexión muy fácil y simplificada, ofrece una serie de servicios, como Autenticación, o pruebas, que otras opciones no ofrecen, y que es justo lo que se necesita en un proyecto de estas características, ya que el trabajo se quiere enfocar principalmente en el desarrollo de la aplicación.

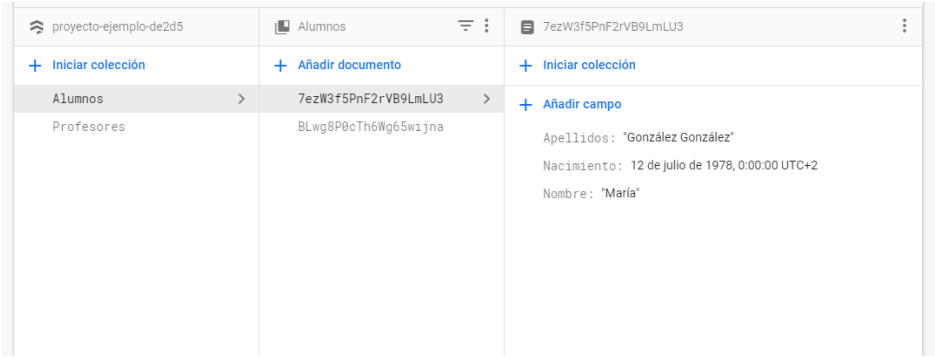


Figura 3.2: Ejemplo ilustrativo de cómo se almacenan los datos en Firestore.

3.2.1 Autenticación

El sistema de autenticación permite el uso de la cuenta de Twitter, Facebook, GitHub, Google o un sistema personalizado basado en la creación de cuentas con nombre de usuario y contraseña. Firebase adjudica un ID único que puede usarse como clave para el perfil privado de cada usuario en la base de datos [12]. Hacer uso del servicio de autenticación no es nada complicado, e incluso la documentación de *Firebase* tiene una guía paso por paso de cómo llevarla a cabo [13].

3.2.2 Base de datos

Para este servicio Firebase nos ofrece dos opciones: *Firestore* [14] y *Real Time Data Base* [15]. Ambos servicios a nivel general son muy parecidos, pero podemos destacar varias de las ventajas de *Firestore* frente a *Real Time Database*. La principal es que se adapta mejor a proyectos que van creciendo o añadiendo nuevas funcionalidades. Además permite añadir ínidices en columnas, lo que posibilita hacer consultas más potentes, con el consecuente ahorro de trabajo en la gestión de datos en la aplicación.

Una vez centrados en *Firestore*, su principal característica, al mismo tiempo que *Real Time Data Base*, es que es una base de datos no relacional. Lo que entenderíamos como tablas, aquí serán colecciones. Y cada una de las filas de una tabla, aquí será un documento en formato JSON. Podemos ver un pequeño ejemplo ilustrativo de estas propiedades en la figura 3.2.

La escritura de datos se lleva a cabo creando un objeto de tipo *HashMap* en el que relacionamos clave-valor, y a partir de una referencia a la base de datos, hacemos un *add(objeto)*, poniendo en el objeto el HashMap creado. En la figura 3.3 vemos un pequeño ejemplo de la creación de un documento en la colección *Profesor*.

La lectura de datos se lleva a cabo realizando una llamada y esperando por su respuesta. En la figura 3.4 tenemos un ejemplo de cómo se traerían los documentos de la colección *Profesor*.

```
val profesor = hashMapOf(
        "Nombre" to "María",
        "Apellidos" to "Domínguez"
)

db.collection("Profesores")
    .add(profesor)
    .addOnSuccessListener { documentReference ->
        Log.d(TAG, "DocumentSnapshot added with ID:
    ${documentReference.id}")
    }
    .addOnFailureListener { e ->
        Log.w(TAG, "Error adding document", e)
    }
```

Figura 3.3: Código de ejemplo de creación en Firestore

```
db.collection("Profesor")
    .get()
    .addOnSuccessListener { result ->
        for (document in result) {
            Log.d(TAG, "${document.id} => ${document.data}")
        }
    }
    .addOnFailureListener { exception ->
        Log.d(TAG, "Error getting documents: ", exception)
    }
```

Figura 3.4: Código de ejemplo de lectura en Firestore

```
var file = Uri.fromFile(File("path/to/images/rivers.jpg"))
val riversRef = storageRef.child("images/${file.lastPathSegment}")
uploadTask = riversRef.putFile(file)

// Register observers to listen for when the download is done or if
    it fails
uploadTask.addOnFailureListener {
    // Caso en el que sucede algún error
}.addOnSuccessListener {
    // Caso en el que se realiza la subida de forma correcta
}
```

Figura 3.5: Código de ejemplo subida de imagen a *Cloud Storage*

```
val ref = storageRef.child("images/mountains.jpg")
uploadTask = ref.putFile(file)

val urlTask = uploadTask.continueWithTask { task ->
    if (!task.isSuccessful) {
        task.exception?.let {
            throw it
        }
    }
    ref.downloadUrl
}.addOnCompleteListener { task ->
    if (task.isSuccessful) {
        val downloadUri = task.result
    } else {
        // Handle failures
        // ...
    }
}
```

Figura 3.6: Código de ejemplo para obtención de *url* para descarga de imagen de *Cloud Storage*

3.2.3 Almacenamiento

El sistema de gestión de almacenamiento de *Firebase*, *Cloud Storage*, da la posibilidad de implementar la gestión de subida y descarga de imágenes de forma sencilla [16]. Se trata, igual que en el caso de *Firestore*, de crear una instancia que haga referencia al servicio de almacenamiento, y a partir de él, podremos o subir una imagen desde un fichero local, como en la figura 3.5, o descargarla de ser necesario, como en el ejemplo de la figura 3.6.

3.3 Librerías para Android

Para la realización de este proyecto se hicieron uso de algunas librerías que fueron de gran utilidad para distintos problemas que su pudieran presentar

- Lifecycle: Los componentes optimizados para ciclos de vida realizan acciones en respuesta a un cambio en el estado del ciclo de vida de otro componente, como actividades o fragmentos. El paquete *androidx.lifecycle* [17] ofrece interfaces y clases que permiten compilar componentes optimizados para ciclos de vida; es decir, componentes que pueden ajustar automáticamente su comportamiento en función del estado actual del ciclo de vida de una actividad o un fragmento. Es una librería nativa de Android que es indispensable a la hora de trabajar con una arquitectura MVVM, arquitectura que se explica con detalle en el apartado 6.1.4.

- Kodein (*KOtlin DEpendency INjection*): Es un framework en Kotlin para inyección de dependencias [18]. Aunque se barajaron otras posibilidades como *Dagger* [19], o *Koin* [20], se decidió optar por esta, por estar implementada en Kotlin y para Kotlin, tener una fácil implementación en android, y una curva de aprenidzaje muy básica a la hora de trabajar con ella.

- AppIntro: Librería que facilita la implementación y gestión de *Intros de aplicación*, haciendo que el desarrollador sólo se tenga que centrar en el apartado de visual de esta [21].

- Material Design: Librería que facilita una serie de componentes visuales que permiten una gran personalización visual y funcional, al mismo que tiempo que aisla de ciertas cuestiones técnicas al desarrollador, llevándolo a tener que centrarse tan sólo en el apartado visual [22].

- Lottie: Lottie es una librería tanto para Andrid, iOS, Web o Windows, que analiza un archivo de animaciones de *Adobe After Effects* y renderiza su animación de forma nativa [23].

3.4 Herramientas de edición y lenguajes de programación

3.4.1 Kotlin

Lenguaje de programación orientado a objetos desarrollado por *JetBrains*, que se adapta a la perfección a distintos IDE's. Busca simplificar la cantidad de código y eliminar los errores más comunes de lenguajes de programación, como Java. Actualmente, Kotlin [24], es el

lenguaje recomendado por Google e incluso fomentado a la hora de desarrollar aplicaciones móviles, y es por ello que, a pesar de no poseer una formación previa por el equipo de desarrollo en este lenguaje, se ha decidido optar por él.

3.4.2 XML

El lenguaje XML [25] es similar a HTML utilizado, en este caso, para construir las vistas, a nivel visual, en aplicaciones móviles.

3.4.3 Android Studio

Android Studio [26] es el entorno de desarrollo integrado (IDE) oficial para el desarrollo de apps para Android, basado en IntelliJ IDEA. Además del editor de código y de las herramientas para desarrolladores de IntelliJ, Android Studio ofrece funciones para aumentar la productividad en cuanto al desarrollo apps para Android.

3.4.4 Trello

Trello [27] es un software de administración de proyectos con interfaz web y con cliente para iOS y android para organizar proyectos. Empleando el sistema kanban, para el registro de actividades con tarjetas virtuales organiza tareas, permite agregar listas, adjuntar archivos, etiquetar eventos, agregar comentarios y compartir tableros.

3.4.5 Figma

Figma [28] es una aplicación para diseñar interfaces que se ejecuta en el navegador. Brinda todas las herramientas necesarias para la fase de diseño del proyecto, incluidas las herramientas vectoriales capaces de ilustrar completamente, así como aquellas para la creación de prototipos y la generación de código para el traspaso (hand-off).

3.4.6 InksCape

Inkscape [29] es un editor de gráficos vectoriales de código abierto, que facilita las herramientas para el dibujo vectorial, necesario a la hora de crear iconos y logos personalizados.

3.4.7 AfterEffects

AfterEffects [30], es una aplicación que tiene forma de estudio destinada a la creación de gráficos profesionales en movimiento y efectos especiales. Para este proyecto se utilizó para generar las animaciones personalizadas.

3.4.8 Overleaf y Latex

Overleaf [31] es una herramienta de publicación colaborativa en línea con un editor LaTeX fácil de usar con colaboración en tiempo real y la salida totalmente compilada producida automáticamente en segundo plano a medida que escribe.

3.4.9 MagicDraw

MagicDraw [32], herramienta CASE(Computer Aided Software Engineering), desarrollada por *No Magic* para la elaboración de diagramas y demás esquemas necesarios en un proyecto de este tipo.

3.4.10 DIA Diagrams

Dia [33] es una aplicación informática de propósito general para la creación de diagramas, concebida de forma modular, con diferentes paquetes de formas para diferentes necesidades.

Capítulo 4

Análisis y requisitos

En este capítulo se profundizará en la fase de análisis del desarrollo del proyecto. Se especificarán los requisitos tanto funcionales como no funcionales, así como los actores y los casos de uso que se extraen de ellos.

4.1 Requisitos funcionales

La lista final de los requisitos funcionales de nuestro sistema es la siguiente:

1. **Acceso funcionalidades**: Gestionar usuarios autenticados y sin autenticar. Los usuarios sin autenticar solo podrán ver la información, y los usuarios autenticados tendrán acceso completo a todas las funcionalidades.
2. **Intercambio de información geolocalizada**: Obtención de la posición del dispositivo móvil para la gestión de la información recuperada.
3. **Puntos de interés geolocalizados**: Creación y edición de puntos de interés.
4. **Valoraciones sobre puntos de interés geolocalizados**: Creación y edición de valoraciones sobre los puntos de interés.
5. **Multimedia**: Posibilidad de asociar imágenes identificativas a los puntos de interés.

4.2 Requisitos no funcionales

Además de los requisitos funcionales, se han planteado una serie de requisitos no funcionales:

- **Arquitectura e implementación generalizable**: Desarrollar una arquitectura y una implementación que sea fácilmente aplicables a otras categorías.

- **Velocidad de respuesta**: Reducir al máximo posible los tiempos de respuesta y que la interacción con el usuario sea ágil y sin retrasos.
- **Interfaz**: La interfaz debe ser estética, rápida e intuitiva.
- **Recursos del sistema**: Tener en cuenta en todo momento la utilización de recursos del dispositivo móvil.

4.3 Actores

En el modelo de casos de uso, los actores representan los roles que desempeñan los distintos usuarios que interactúan con la aplicación. Para la tenemos los siguientes actores:

- **Administrador**: Es el usuario que se encarga de la gestión del sistema, tanto de la aplicación, como de la base datos, así como de Firebase.
- **Usuario sin autenticar**: Usuario que interactúa con la aplicación. Este usuario sólo podrá acceder a la información almacenada, en ningún caso podrá crear nuevas localizaciones o valoraciones.
- **Usuario Autenticado**: Extensión del usuario sin autenticar. Este, aparte de poder ver los datos, podrá crear localizaciones y valoraciones, como también editar y eliminar aquellas que haya creado.
- **Usuario Moderador**: El administrador podrá dar de alta usuarios moderadores. Estos usuarios son una extensión del Usuario Autenticado, pero en este caso, podrán editar y/o eliminar cualquier localización o valoración sin ningún tipo de impedimento.

4.4 Casos de uso

A continuación tenemos los casos de uso que puede llevar a cabo cada uno de los actores [34]. En el apéndice B se muestran todos los detalles de cada uno de ellos.

4.4.1 Casos de uso *Usuario sin autenticar*

CU-01 Autenticarse: El usuario podrá identificarse a través de un token. Si es llevado a cabo, su rol pasará a ser *Usuario autenticado.* (Tabla B.1)

CU-02 Cargar localizaciones cercanos: A partir de su posición actual, el usuario podrá ver en un mapa la información de las localizaciones cercanas almacenadas en la bas de datos. Para evitar sobrecargas, sólo se cargan las localizaciones que se encuentren en la

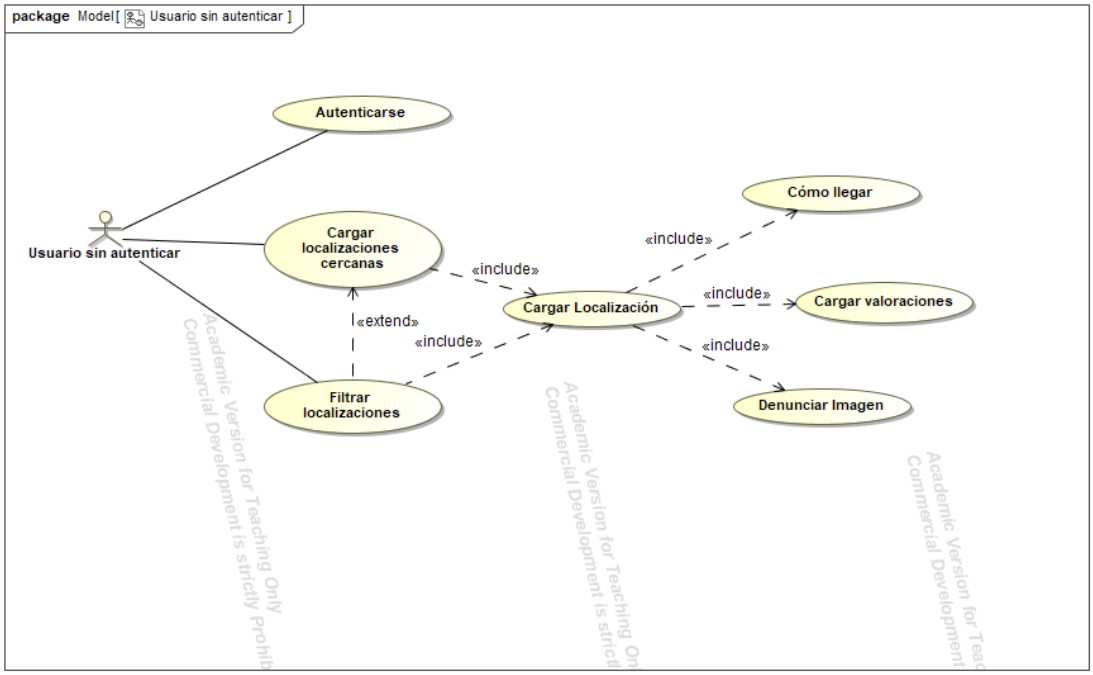


Figura 4.1: Casos de uso del usuario sin autenticar

misma ciudad y país, y tan sólo las que se encuentran en un radio de 500 metros a la redonda. (Tabla B.2)

CU-03 Buscar localizaciones: El usuario podrá seleccionar una serie de campos para que sólo se muestren las localizaciones que se correspondan con ellos. (Tabla B.3)

CU-04 Cargar localización: Al seleccionar una localización, se verá la información relacionada con ella, y las medias y demás datos calculados a partir de las valoraciones, sobre la localización. (Tabla B.4)

CU-05 Denunciar Imagen: Los usuarios podrán denunciar imágenes que sean inapropiadas o incumpla alguna normativa. Se realizará un borrado lógico y un aviso al administrador. (Tabla B.5)

CU-06 Cargar Valoraciones: El usuario podrá acceder a las valoraciones relacionadas a una localización. Por simplificar la cantidad de información que verá el suario, sólo se mostrarán los comentarios y notas generales de cada una de las valoraciones, así como también qué otro usuario la realizó. (Tabla B.6)

CU-08 Cómo Llegar: A partir de una localización, se facilitará la información sobre cómo llegar a ella a partir de la posición actual del usuario. (Tabla B.8)

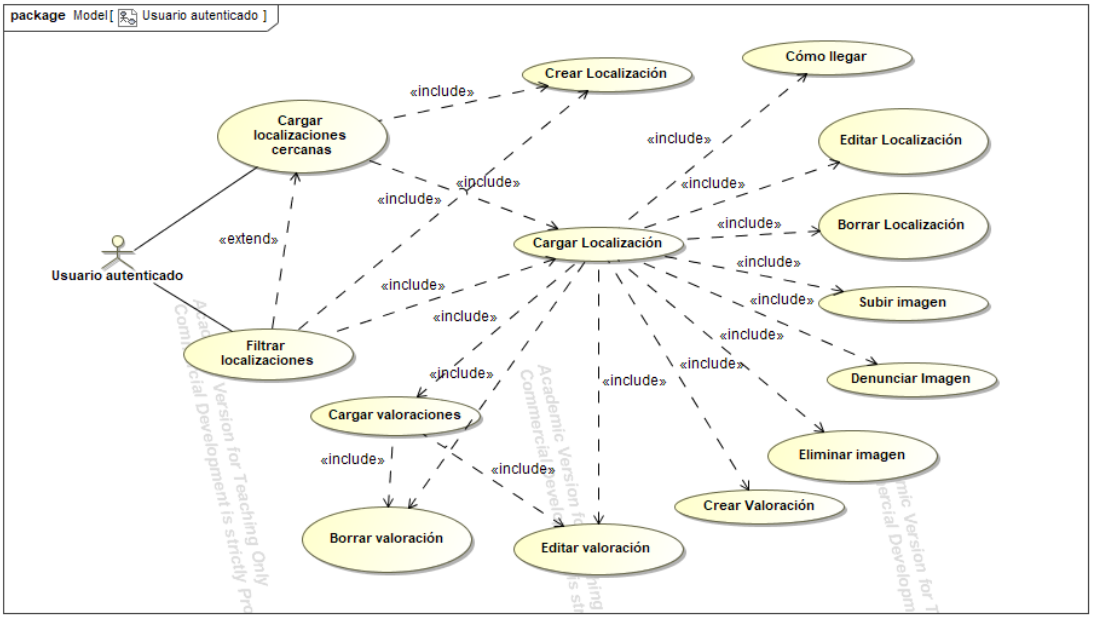


Figura 4.2: Casos de uso del usuario autenticado

4.4.2 Casos de uso *Usuario autenticado*

Este rol es una extensión del *Usuarion sin autenticar*, por lo que podrá llevar a cabo los mismos casos de uso, pero a mayores podrá realizar algunos más, una cuestión que se puede ver a simple vista si comparamos las figuras 4.1 y 4.2.

CU-07 Crear Localización: A partir de su posición actual, el usuario podrá crear una localización nueva. Sólo será necesario aportar datos de nombre y descripción. También, por evitar un mal uso de la aplicación, como crear localizaciones en ciudades en las que no se ha estado, y ser fiel a la filosofía de estar hecha para el uso en los trayectos cotidianos, sólo se permitirá crear en puntos que no se encuentren a una distancia mayor de 500 metros de la posición actual del usuario. (Tabla B.7)

CU-09 Editar Localización: A partir de una localización seleccionada, si el usuario autenticado es el creador de la misma, podrá editar su nombre y/o descripción. Sólo se permitirá la edición de estos campos, ya que el resto de campos serán calculados a partir de las valoraciones. (Tabla B.9)

CU-10 Borrar Localización: A partir de una localización seleccionada, si el usuario autenticado es el creador de la misma, podrá eliminar esta localización. Si una localización es eliminada, también se eliminarán todas las valoraciones asociadas. (Tabla B.10)

CU-11 Subir Imagen: Habiendo seleccionado una localización, el usuario podrá subir una

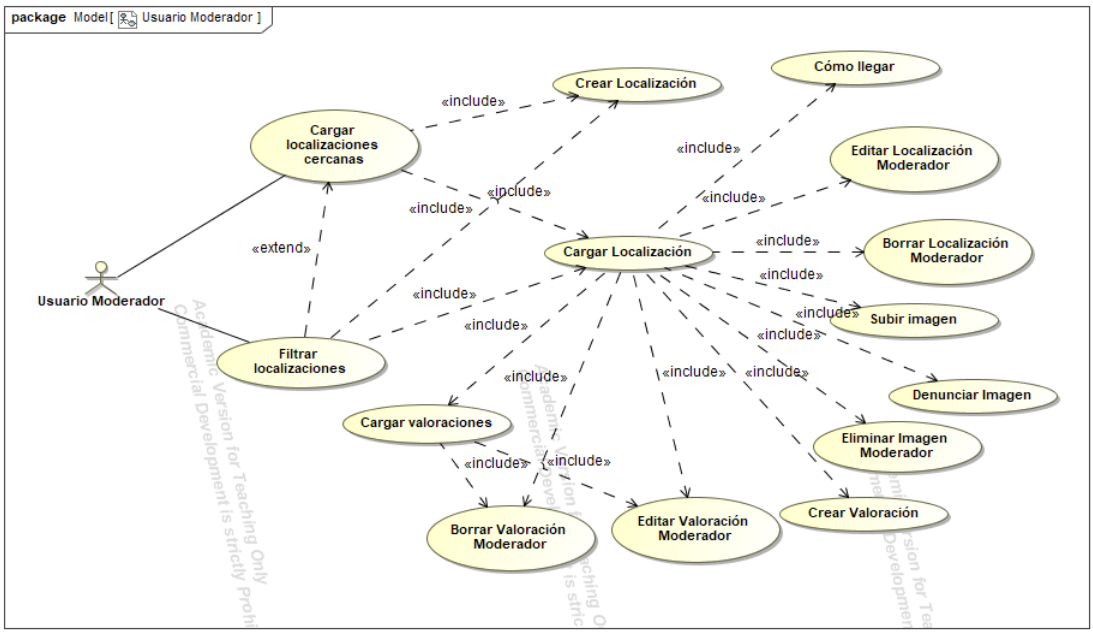


Figura 4.3: Casos de uso del usuario Moderador

imagen sobre ella. (Tabla B.11)

CU-12 Eliminar Imagen: Si el usuario es el que ha subido una imagen, podrá eliminarla. (Tabla B.12)

CU-13 Crear Valoración: A partir de una localización, el usuario podrá crear una valoración. Las valoraciones constarán de una serie de parámetros que medirán la experiencia de los usuarios, y que se usarán para calcular la información que se enseñará sobre la localización. (Tabla B.13)

CU-14 Editar Valoración: Si el usuario es el creador de una valoración, podrá editar el comentario de ella. El resto de parámetros, al ser parámetros fijos y que en gran medida dependen del momento, no se podrán editar, si se quiere meter nuevos valores para estos parámetros, se tendrá que crear una nueva valoración. (Tabla B.14)

CU-15 Borrar Valoración: Si el usuario es el creador de una valoración, podrá eliminarla. (Tabla B.15)

4.4.3 Casos de uso *Usuario Moderador*

Como sucedía con el *Usuario autenticado*, el *Usuario Moderador* es una extensión del *Usuario sin autenticar*, por lo que podrá llevar a cabo los mismos casos de uso, pero a mayores podrá realizar otros, cuestión que se puede ver en la figura 4.3.

CU-16 Editar Localización Moderador: Su funcionalidad será la misma que el caso de uso *Editar Localización* (Tabla B.9), pero en este caso, este usuario no tendrá la restricción de tener que haber sido el creador de la localización, sino que podrá eliminar cualquier localización. (Tabla B.16)

CU-17 Borrar Localización Moderador: Al igual que el caso de uso CU-16, este caso de uso es una extensión de un caso de uso del *Usuario autenticado*, por lo que podrá eliminar localizaciones sin ningún tipo de restricción. (Tabla B.17)

CU-18 Borrar Imagen Moderador: Se repite la misma casuística que en los casos de uso anteriores, por lo que este usuario podrá borrar una imagen sin tener que ser quien la ha subido. (Tabla B.18)

CU-19 Editar Valoración Moderador: El usuario moderador podrá editar el comentario de una valoración sin restricciones. (Tabla B.19)

CU-20 Borrar Valoración Moderador: El usuario podrá eliminar una valoración sin tener que ser su creador. (Tabla B.20)

Capítulo 5

Metodología de desarrollo

En este capítulo se describen las propiedades más relevantes de las metodologías de desarrollo ágiles y por qué consideramos que son apropiadas para este proyecto. A continuación, se explica en más detalle la metodología seleccionada, *Extremme Programming (XP)*. Por último, se comentan las principales modificaciones aplicadas a XP debido a las características específicas del presente proyecto [35].

5.1 Metodologías ágiles

Las metodologías denominadas ágiles se basan en el desarrollo iterativo e incremental, donde los requisitos y soluciones evolucionan mediante la colaboración.

Estas metodologías se basan en los siguientes principios:

- Satisfacer al cliente a través de entregas continuas y tempranas es la mayor prioridad.
- Los cambios a los requerimientos son bienvenidos, aún en fases tardías del desarrollo.
- Entregar frecuentemente software que funciona, desde un par de semanas a un par de meses, prefiriendo los periodos más cortos.
- Desarrolladores, gerentes y clientes deben trabajar juntos diariamente, a lo largo del proyecto.
- Construir proyectos alrededor de personas motivadas, dándoles el entorno y soporte que necesitan, y confiando en que realizarán el trabajo.
- El método más eficiente y efectivo de transmitir información entre un equipo de desarrolladores es la conversación frontal (cara a cara).
- Tener software que funciona es la medida primaria del progreso.

- El proceso ágil promueve el desarrollo sostenible. Los sponsors, desarrolladores y usuarios deben ser capaces de mantener un ritmo de trabajo constante en forma permanente a lo largo del proyecto.

- La atención continua a la excelencia técnica y el buen diseño mejoran la agilidad.

- Simplicidad – el arte de maximizar el trabajo que no se debe hacer – es esencial.

- Las mejores arquitecturas, requerimientos y diseños surgen de los equipos auto-organizados.

- A intervalos regulares, el equipo debe reflexionar sobre como ser más efectivos, y ajustar su comportamiento de acuerdo a ello.

5.2 *Extremme Programming* (XP)

La metodología XP [35] define cuatro variables para cualquier proyecto de software: costo, tiempo, calidad y alcance. Además, se especifica que, de estas cuatro variables, sólo tres de ellas podrán ser fijadas arbitrariamente por actores externos al grupo de desarrolladores (clientes y jefes de proyecto). El valor de la variable restante podrá ser establecido por el equipo de desarrollo, en función de los valores de las otras tres.

El ciclo de vida de un proyecto XP incluye, al igual que las otras metodologías, entender lo que el cliente necesita, estimar el esfuerzo, crear la solución y entregar el producto final al cliente. Sin embargo, XP propone un ciclo de vida dinámico, donde se admite expresamente que, en muchos casos, los clientes no son capaces de especificar sus requerimientos al comienzo de un proyecto.

Por esto, se trata de realizar ciclos de desarrollo cortos (llamados iteraciones), con entregables funcionales al finalizar cada ciclo. En cada iteración se realiza un ciclo completo de análisis, diseño, desarrollo y pruebas, pero utilizando un conjunto de reglas y prácticas que caracterizan a XP (y que serán detalladas más adelante).

En comparación a otras metdologías ágiles, *Extremme Programming* trabaja en iteraciones más cortas, permite cambios durante el desarrollo de un *sprint*, tiene una serie de *buenas prácticas* que pueden ayudar a lo largo del desarrollo y no necesita, como en *Scrum*, de un *Scrum Master* que vaya priorizando tareas, sino que se prioriza en un principio y se sigue al pie de la letra esta priorización. Teniendo en cuenta estas cuestiones y el nivel experiencia inicial del equipo de desarrollo con muchas de las tecnologías a utilizar y que el equipo desarrollo está conformado por una sola persona, *Extremme Programming* es la que mejor se adapta al proyecto en cuestión, y por ello se escogió.

5.2.1 Fases

Si bien el ciclo de vida de un proyecto XP es muy dinámico, se puede separar en fases

- Fase de exploración: Es la fase en la que se define el alcance general del proyecto. En esta fase, el cliente define lo que necesita mediante la redacción de sencillas "historias de usuarios". Los programadores estiman los tiempos de desarrollo en base a esta información.

- Fase de planificación: La planificación es una fase corta, en la que el cliente, los gerentes y el grupo de desarrolladores acuerdan el orden en que deberán implementarse las historias de usuario, y, asociadas a éstas, las entregas.

- Fase de iteraciones: Esta es la fase principal en el ciclo de desarrollo de XP. Las funcionalidades son desarrolladas en esta fase, generando al final de cada una un entregable funcional que implementa las historias de usuario asignadas a la iteración. Como las historias de usuario no tienen suficiente detalle como para permitir su análisis y desarrollo, al principio de cada iteración se realizan las tareas necesarias de análisis, recabando con el cliente todos los datos que sean necesarios.

- Fase de puesta en producción: Si bien al final de cada iteración se entregan módulos funcionales y sin errores, puede ser deseable por parte del cliente no poner el sistema en producción hasta tanto no se tenga la funcionalidad completa. En esta fase no se realizan más desarrollos funcionales, pero pueden ser necesarias tareas de ajuste ("fine tuning").

5.2.2 Reglas y prácticas

La metodología XP tiene un conjunto importante de reglas y prácticas.

- **Historias de usuarios**
  Descripciones cortas de lo que el sistema debe realizar.Las historias de usuario deben tener el detalle mínimo como para que los programadores puedan realizar una estimación poco riesgosa del tiempo que llevará su desarrollo.

- **Plan de entregas ("Release Plan")**
  El cronograma de entregas establece qué historias de usuario serán agrupadas para conformar una entrega, y el orden de las mismas. Este cronograma será el resultado de una reunión entre todos los actores del proyecto (cliente, desarrolladores, gerentes, etc.).

- **Plan de iteraciones ("Iteration Plan")**
  Las historias de usuarios seleccionadas para cada entrega son desarrolladas y probadas en un ciclo de iteración, de acuerdo al orden preestablecido. Al comienzo de cada ciclo, se realiza una reunión de planificación de la iteración. Cada historia de usuario se traduce en tareas especificas de programación.

- **Reuniones diarias de seguimiento ("Stand-up meeting")**
  Reuniones diarias para mantener la comunicación entre el equipo, y compartir problemas y soluciones.

- **Simplicidad**
  Implementar el diseño más simple posible que funcione. Se sugiere nunca adelantar la implementación de funcionalidades que no correspondan a la iteración en la que se esté trabajando.

- **Soluciones "spike"**
  Cuando aparecen problemas técnicos, o cuando es difícil de estimar el tiempo para implementar una historia de usuario, pueden utilizarse pequeños programas de prueba (llamados "spike"1), para explorar diferentes soluciones. Estos programas son únicamente para probar o evaluar una solución, y suelen ser desechados luego de su evaluación.

- **Recodificación**
  La recodificación ("refactoring") consiste en escribir nuevamente parte del código de un programa, sin cambiar su funcionalidad, a los efectos de hacerlo más simple, conciso y/o entendible. Muchas veces, al terminar de escribir un código de programa, pensamos que, si lo comenzáramos de nuevo, lo hubiéramos hecho en forma diferente, mas clara y eficientemente. Sin embargo, como ya está pronto y "funciona", rara vez es reescrito. Las metodologías de XP sugieren recodificar cada vez que sea necesario. La filosofía que se persigue es, como ya se mencionó, tratar de mantener el código más simple posible que implemente la funcionalidad deseada.

- **Metáforas**
  Una "metáfora" es algo que todos entienden, sin necesidad de mayores explicaciones. La metodología XP sugiere utilizar este concepto como una manera sencilla de explicar el propósito del proyecto, y guiar la estructura y arquitectura del mismo. Por ejemplo, puede ser una guía para la nomenclatura de los métodos y las clases utilizadas en el diseño del código. Tener nombres claros, que no requieran de mayores explicaciones, redunda en un ahorro de tiempo.

- **Disponibilidad del cliente**
  Uno de los requerimientos de XP es tener al cliente disponible durante todo el proyecto. No solamente como apoyo a los desarrolladores, sino formando parte del grupo. El involucramiento del cliente es fundamental para que pueda desarrollarse un proyecto con la metodología XP.

- **Uso de estándares**
  Si bien esto no es una idea nueva, XP promueve la programación basada en estándares, de manera que sea fácilmente entendible por todo el equipo, y que facilite la recodificación.

- **Programación dirigida por las pruebas ("Test-driven programming")**
  En las metodologías tradicionales, la fase de pruebas, incluyendo la definición de los tests, es usualmente realizada sobre el final del proyecto, o sobre el final del desarrollo de cada módulo. La metodología XP propone un modelo inverso, en el que, lo primero que se escribe son los test que el sistema debe pasar. Luego, el desarrollo debe ser el mínimo necesario para pasar las pruebas previamente definidas.

- **Programación en pares**
  XP propone que se desarrolle en pares de programadores, ambos trabajando juntos en un mismo ordenador. Si bien parece que ésta práctica duplica el tiempo asignado al proyecto (y por ende, los costos en recursos humanos), al trabajar en pares se minimizan los errores y se logran mejores diseños, compensando la inversión en horas.

- **Integraciones permanentes**
  Todos los desarrolladores necesitan trabajar siempre con la "última versión".

- **Propiedad colectiva del código**
  En un proyecto XP, todo el equipo puede contribuir con nuevas ideas que apliquen a cualquier parte del proyecto. Asimismo, cualquier pareja de programadores puede cambiar el código que sea necesario para corregir problemas, agregar funciones o recodificar.

- **Ritmo sostenido**
  La metodología XP indica que debe llevarse un ritmo sostenido de trabajo. El concepto que se desea establecer con esta práctica es el de planificar el trabajo de manera de mantener un ritmo constante y razonable, sin sobrecargar al equipo.

5.3 Aplicación de XP en este proyecto

Se ha elegido XP primeramente por tratarse de una metodología ágil y flexible, y que en comparación a otras metodologías ágiles como Scrum, permite un trabajo mucho más flexible, cuestión importante en el proyecto al que se va a aplicar, dado que el equipo de desarrollo está formado por una sola persona. Además, se emplearán tecnologías de las que el desarrollador tiene un conocimiento superficial. Por lo tanto, la curva de aprendizaje y, sobre todo, la estimación de tiempos y complejidad inicialmente no será demasiado fiable. Por todo ello, la metodología de desarrollo tiene que ser flexible.

Si comprobamos las reglas y prácticas que usa XP en el desarrollo, podemos comprobar como hay varias cuestiones que habrá que adaptar para un correcto funcionamiento de la metodología en este proyecto:

- Las reuniones diarias se limitarán a una actualización diaria del backlog de tareas e historias de cada sprint.
- Dado el tiempo ajustado desde que se comienza el proyecto hasta la fecha de fin, las *pruebas unitarias* se omitirán, y en cambio se realizará al principio de cada *sprint* un documento detallado de cómo debe ser el funcionamiento de la tarea desarrollada, para que al final del mismo, en las *pruebas de caja negra*, pruebas que explicaremos en detalle en el apartado 9.1.1, se tenga una guía detallada de qué y cómo debe funcionar.
- Las programación en pares se omitirá al ser sólo un desarrollador al mismo tiempo que la propiedad colectiva del código.
- La disponibilidad constante del cliente, al hacer de cliente entre el director del trabajo de fin de grado y el desarrollador, se sobreentiende que se cumplirá sin problemas.

El resto de reglas y prácticas se mantendrán en mayor medida como aparece indicado en el subapartado anterior.

Capítulo 6

Arquitectura

En este capítulo explicaremos la arquitectura general del sistema y cada uno de los componentes que lo conforman. Además, describiremos el modelo de datos empleado.

6.1 Arquitectura del sistema

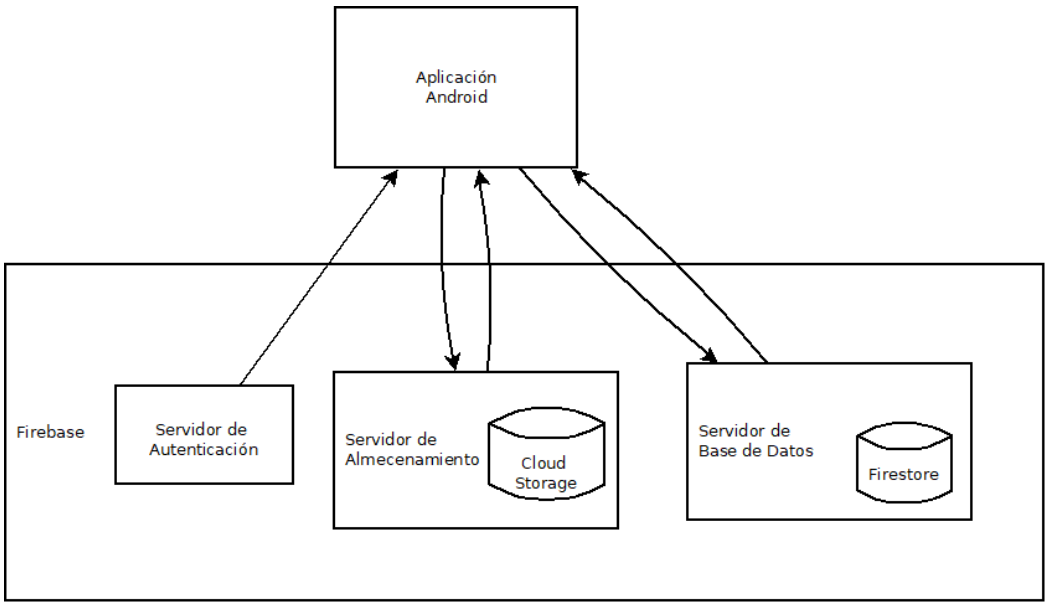


Figura 6.1: Esquema general de la arquitectura del sistema propuesto.

En la figura 6.1 podemos ver la arquitectura general del sistema. Los componentes que la conforman son:

- Aplicación Android:
  Componente central del sistema y en el que se centró la mayor parte del desarrollo. Se explica con más detalle en el apartado 6.1.4.

- Servidor de autenticación:
  El sistema necesita poder autenticar a los usuarios que así lo deseen, y saber al mismo tiempo qué usuarios no están autenticados. Existen dos opciones: implementar un servidor propio de autenticación, con registros, usuarios y contraseñas; o utilizar un servicio de autenticación externo (Google, Twitter, Facebook, etc...).
- Servidor de la Base de datos:
  Base de datos en la nube que permite a la aplicación Android acceder a los datos disponibles, así como actualizarlos.
- Servidor de almacenamiento:
  Teniendo en cuenta el caso de uso B.10, se hace necesario que el sistema pueda almacenar imágenes y es deseable que no se integren en la base de datos.

A continuación explicaremos un poco más en detalle cada uno de los componentes.

6.1.1 Autenticación

Firebase ofrece un sistema de autenticación totalmente integrado con la base de datos. Permite el uso de un sistema de autenticación personalizado con nombre de usuario y contraseña, o la autenticación por medio de algún proveedor de entidades federadas como Google, Twitter, Facebook y Github. Utilizar la autenticación por medio de Firebase mediante proveedor de entidades federadas nos ofrece una solución rápida y práctica a la vez que es fácilmente sustituible por un sistema propio si fuese necesario.

Por otra parte, todos los móviles Android deben tener al menos una cuenta de Google asociada al dispositivo, por lo que no es necesaria la creación de ninguna cuenta adicional. Además Android integra el manejo de cuentas de Google y hace muy cómodo cambiar entre ellas. La aplicación puede pedir un *token* al servidor de autenticación, y a partir de él ya identifica al usuario.

6.1.2 Base de datos

Este componente también estará basado en un servicio que ofrece *Firebase*, en este caso será *Firestore*, que nos permite la implementación de una base de datos y la conexión a ella de una forma muy fácil y sencilla. Además de ello, da la posibilidad de gestionar datos en tiempo real, ya que los cambios a los que se ve sometida la base de datos, se ven reflejados con inmediatez.

Otra de las grandes ventajas de *Firestore* es la escalabilidad que posee en caso de que el proyecto en poco tiempo adquiera un tamaño mayor al planificado inicialmente. La aplicación se comunicará directamente con *Firestore*, editando, creando y/o borrando los datos que fuesen necesarios.

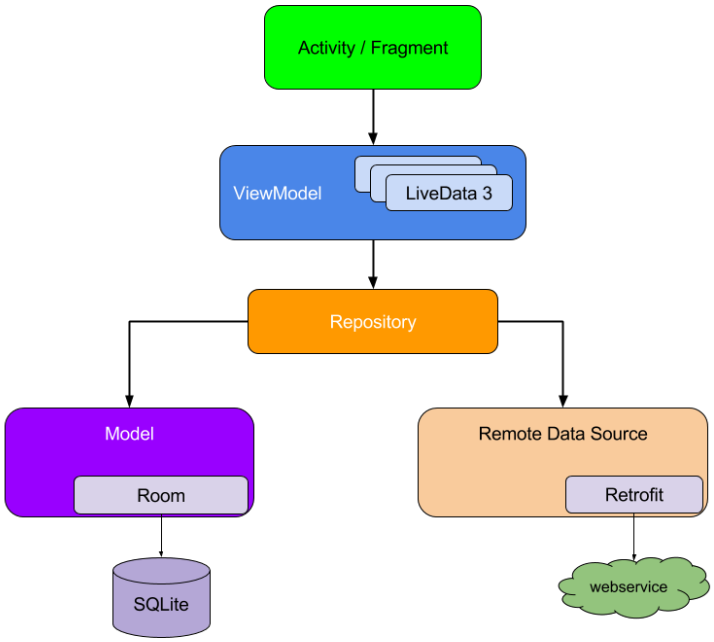


Figura 6.2: Esquema general de una arquitectura *Model-View-View-Model* (MVVM).

6.1.3 Servidor de almacenamiento

Dada su facilidad a la hora de trabajar de forma conjunta con el resto de componentes, también se ha utilizado un servicio de *Firebase* para la implementación del servidor de almacenamiento, que también nos facilitará una solución rápida y práctica a la hora de implementar el almacenamiento de grandes volúmenes de datos (por ejemplo, imágenes).

Una de las cuestiones que conlleva el usar *Cloud Storage*, es que es un servicio separado de la base de datos, es decir de *Firestore*, por lo que es muy importante mantener una referencia a los datos almacenados en *Cloud Storage* en la base datos. Así, externamente da la impresión de que ambos servidores están unificados, pero realmente, internamente se hacen distintas peticiones a servicios diferenciados.

6.1.4 Aplicación Android

Para esta aplicación se hizo uso de la la arquitectura recomendada por la documentación de Android, la arquitectura *Model-View-View-Model* [7], de la cual podemos ver un ejemplo de implementación en la figura 6.2. Esta arquitectura se basa en dos principios básicos:

- **Separación de problemas**
  División del problema en distintas subsecciones y una delimitación de responsabilidades lo más clara posible.

- **Controlar la IU a partir de un modelo**
  No controlar la IU a partir a partir de la vista, sino que ceder esta responsabilidad a un modelo de datos persistente.

Para respetar estos principios, la aplicación desarrollada se divide en los siguientes componentes:

- **View**: Es el componente encargado de la interacción con el usuario. Recibe los datos del *View-Model*, y a partir de él realiza los cambios y gestiona los eventos de la interfaz.
- **View-Model**: proporciona los datos para un componente de IU específico, como un fragmento o una actividad. También incluye la lógica empresarial de manejo de datos para comunicarse con el modelo. No tiene conocimiento sobre los componentes de IU, de manera que no se ve afectado por los cambios de configuración
- **Model**: Representa el modelo de dominio de la aplicación, dentro del cual se encuentran el modelo de datos y la lógica empresarial.

6.2 Modelo de datos

En la figura 6.3 podemos ver el modelo entidad relación del sistema. A continuación se explican las tres entidades desarrolladas y sus atributos: *Localización*, *Valoración* e *Imagen*.

6.2.1 Entidad *Localización*

La entidad **Localización** representa los sitios geolocalizados que contienen la información creada por los usuarios. Los atributos de esta entidad son los siguientes:

- ***Id***: Código identificativo unívoco.
- ***Nombre***: Nombre asociado a la localización
- ***Descripción***: Pequeño texto que facilita información sobre la localización (Cafetería, Cine, 1º planta centro comercial, etc...).
- ***Ciudad*** y ***Pais***: Información sobre la localización para facilitar las consultas y evitar respuestas demasiado grandes.
- ***Latitud*** y ***Longitud***: Datos para la geolocalización.
- ***CreadoPor***: Usuario que ha creado la localización.

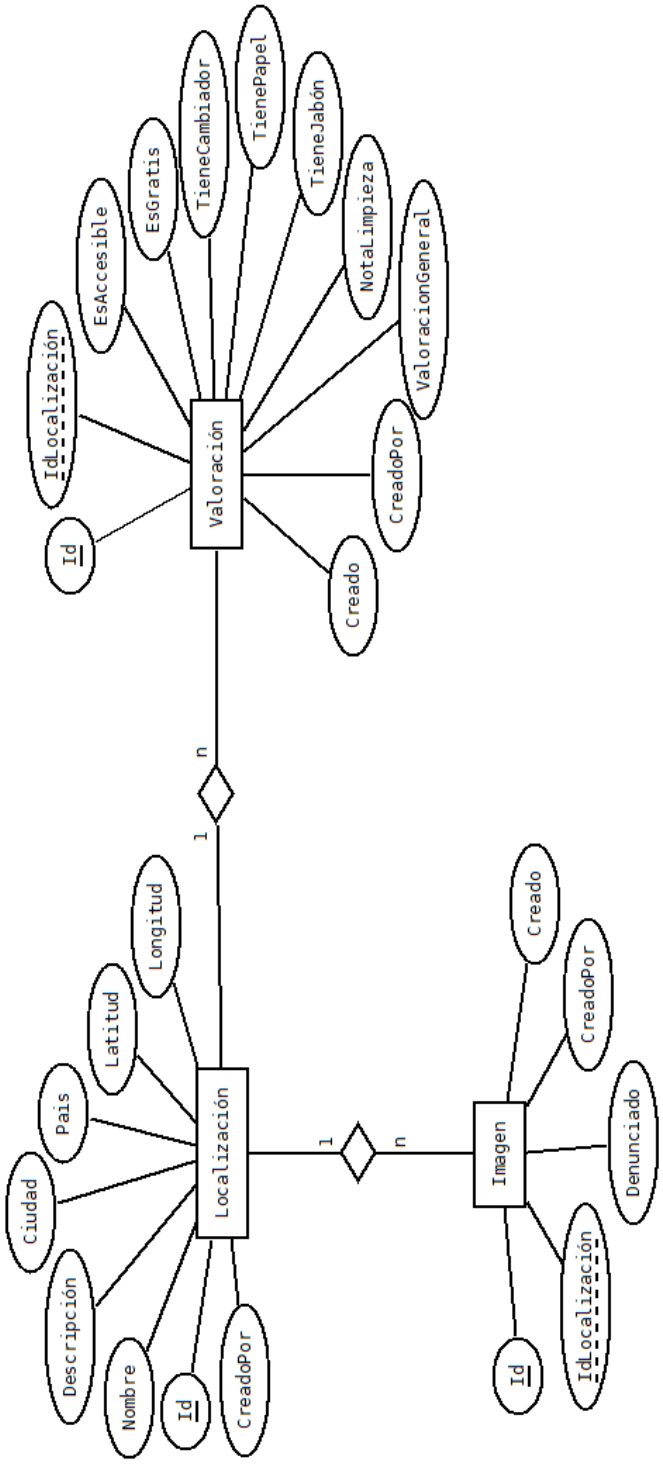


Figura 6.3: Modelo Entidad Relación de nuestra aplicación.

6.2.2 Entidad *Valoración*

Esta entidad es la que guardará toda la información respecto a las experiencias de los usuarios en las localizaciones. La información resumen que se mostrará de cada *Localización* se calculará a partir de los datos almacenados en esta entidad.
Atributos:

- ***Id***: Código identificativo unívoco.
- ***IdLocalizacion***: Código que hace referencia a una *Localizacion*
- Atributos ***Es...*** y ***Tiene...***: Valores booleanos que hacen referencia a distintos aspectos, que se entendieron importantes, sobre la *Localización*. Estos atributos caracterizan a este Modelo de Datos como uno hecho para la valoración de baños geolocalizados, es decir, al pasar este Modelo de Datos a otro tipo de Sistema que se base en la misma filosofía colaborativa de valoración de puntos geolocalizados, son los atributos que tendríamos que cambiar, dejando el resto del Modelo de Datos exactamente igual.
- ***NotaLimpieza***: Valoración númerica de la limpieza de la *Localización*. El único atributo que no es de los anteriores que también deberíamos cambiar en caso de querer aplicar este Modelo de Datos a otro tipo de problema.
- ***ValoracionGeneral***: Nota numérica que valora de forma total a la *Localización*.
- ***Creado***: Fecha de creación.
- ***CreadoPor***: Usuario que la ha creado.

6.2.3 Entidad *Imagen*

Entidad que almacenará la información al respecto de las imágenes subidas al sistema. Sus atributos son:

- ***Id***: Código identificativo unívoco.
- ***IdLocalizacion***: Código que hace referencia a una *Localizacion*
- ***Denunciado***: Valor booleano que hace referencia a si la imagen subida es inapropiada o no. En caso de serlo no se enseñará pero se mantendrá en la base de datos a la espera de la confirmación del administrador.
- ***Creado***: Fecha de creación.
- ***CreadoPor***: Usuario que la ha creado.

Capítulo 7

Implementación

En este capítulo se describen los principales detalles de la implementación realizada. Dedicaremos especial atención a los mecanismos de autenticación implementados en nuestro sistema, a los detalles de la base de datos y a la aplicación Android propiamente dicha.

7.1 Autenticación

Para este apartadado se hizo uso del servicio de autenticación que ofrece Firebase, usando las credenciales de Google, explicado en el apartado 6.1.1. En el momento de iniciar sesión, se abre un pequeño diálogo dando las opciones de cuentas con las que ya se ha iniciado sesión, o hacerlo con una nueva cuenta. Al seleccionar una cuenta, ésta se guarda en el apartado de *Authentication* de *Firebase*, junto con la dirección de correo, la fecha de creación, la fecha del último inicio de sesión y un *uid* único.

En la aplicación hacemos uso del *token* que nos devuelve el servidor de autenticación. Y lo usamos tanto para enseñar la información del usuario autenticado (sus datos de perfil) como para controlar y guardar las acciones que el usuario haya llevado a cabo (por ejemplo, crear una localización, una valoración, etc...). Por lo tanto, nos interesan los siguientes datos:

- *Nombre*: Nombre asociado a la cuenta que se usará para enseñar quién creó una valoración y mantener informado al usuario de qué cuenta inició sesión.
- *Email*: Dirección de correo electrónico asociada a la cuenta, también se usará a la hora de mostrar información.
- *Uid*: Identificador unívoco que se usa para contrastar si el usuario creó o no una valoración o una localización y a la hora de editar/borrar.
- *PhotoUrl*: Dirección Url de la imagen asociada al usuario, que se usará para enseñar la información del usuario que ha iniciado sesión.

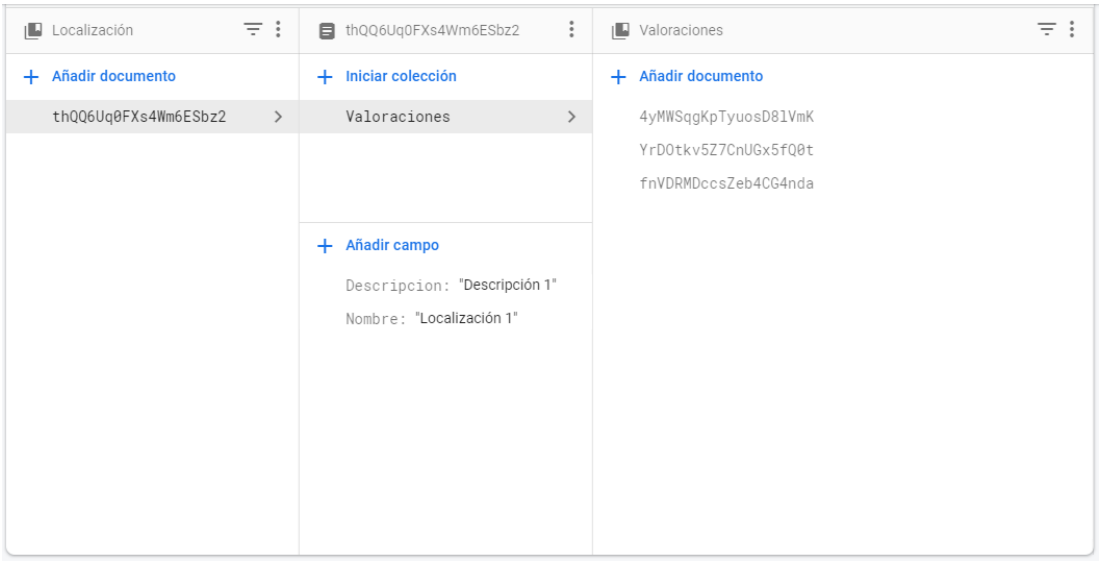


Figura 7.1: Ejemplo de implementación con subcolecciones.

Esta información no se guardará en ninguna parte más allá de la sección *Authenticacion* de *Firebase*. Sólo se mantendrá una relación en los *CreadoPor* de las entidades, y al resto se accederá a través de la instancia creada a *Firebase*. Mientras el usuario se mantenga logueado en una sesión, esa información siempre estará accesible.

7.2 Base de datos

Como *Firestore* es una base de datos no relacional que no trabaja con tablas sino con colecciones de documentos, a la hora de implementar nuestra base de datos se tuvo que prestar atención a una serie de cuestiones muy importantes. En primer lugar, cómo evitar los datos anidados. En segundo lugar, como implementar las consultas de agregación

7.2.1 Evitar datos anidados

Por cómo está definido el modelo entidad relación (apartado 6.2), la entidad *Valoracion* tiene una *clave foránea* que hace referencia a una *localización* de la entidad *Localización*.

Habría dos formas de enfocar este problema:

- Cada localización tendría una subcolección con la lista de valoraciones asociada a ella. Un ejemplo lo podríamos ver en la figura 7.1.
- Crear una colección aparte de *Localizacion* con todas las valoraciones, y que tengan un Id que haga referencia a la localización a modo de *clave foránea*. Como en el ejemplo de la figura 7.2.

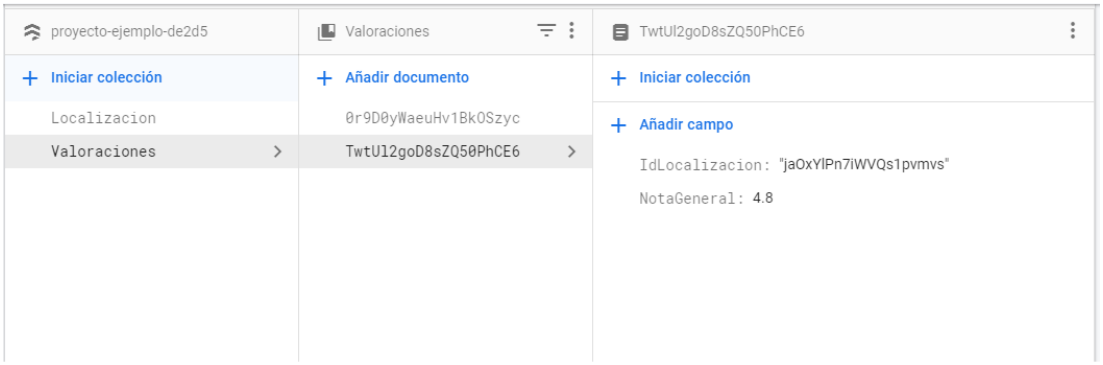


Figura 7.2: Ejemplo de implementación de dos colecciones diferentes con una *clave foránea*.

El principal problema de la primera opción, es que a la hora de querer hacer referencia a una sola *valoración*, ya sea para editarla, cargar sus datos o eliminarla, tendríamos que saber siempre en qué *Localización*. Aunque no debería suponer mayor problema, es un paso que se puede omitir fácilmente aplicando la segunda opción. Es por este motivo por lo que se ha optado por implementar la segunda opción planteada.

7.2.2 Consultas de agregación

En lo relativo a enseñar la información asociada a cada *Localización*, los datos que se muestran, (*valoracion media*, si tiene o no tiene papel higiénico, etc...), son valores calculados a partir de las distintas valoraciones. Si se emplease una base de datos relacional, se trabajaría usando consultas de agregación [36].

Pero, como ya se ha comentado otras veces, *Firestore* no es una base de datos relacional. Para estos casos, la documentación recomienda realizar un cambio en cómo se guardan los datos dejando el cálculo de estos valores al *cliente* [37], y almacenándolos en un campo a mayores en el documento.

Al tratarse de valores que pueden ir cambiando con cada ejecución, se han añadido los siguientes campos a la entidad de *Localizacion*:

- ***numValoraciones***: El número total de valoraciones que posee la *Localizacion*.
- ***totalValoracion*** y ***totalLimpieza***: La suma total del campo *ValoracionGeneral* y *NotaLimpieza* de cada una de las valoraciones. Para calcular la media, en el *cliente* se dividen estos campos por *numValoraciones*.
- ***total...*** : Para el resto de valores booleanos de las valoraciones se ha agregado un campo

```
enum class LocationTableEnum(val colName: String) {
    NOMBRE_COL("nombre"),
    DESCRIPCION_COL("descripcion"),
    CIUDAD_COL("ciudad"),
    PAIS_COL("pais"),
    LATITUD_COL("latitud"),
    LONGITUD_COL("longitud"),
    CREADOPOR_COL("creadoPor"),
    NUMVALORACIONES_COL("numValoraciones"),
    VALORACION_COL("totalValoracion"),
    ACCESIBLE_COL("totalAccesible"),
    ALEATORIO_COL("totalAleatorio"),
    CAMBIADOR_COL("totalCambiador"),
    JABON_COL("totalJabon"),
    LIMPIEZA_COL("totalNotaLimpieza"),
    GRATIS_COL("totalGratis"),
    PAPEL_COL("totalPapel")
}
```

Figura 7.3: Enumerado de la colección ***Localizacion***.

que es el sumatorio de las veces que está a *true*. De nuevo, para calcular el valor que se muestra en la aplicación, se pueden divide el sumatorio en cuestión por el *numValoraciones*, si es mayor que el 50% de las valoraciones, se mostrará a *true*.

7.3 Implementación de la aplicación Android

Tal y como se ha explicado en el apartado 6.1.4, la aplicación Android de nuestro sistema sigue las pautas de la arquitectura *Model-View-View-Model* (MVVM). A continuación explicaremos con más detalle cómo se ha implementado cada capa de la misma: *Model*, *View-Model* y *View*. Por último, dedicaremos un apartado específico a explicar la interfaz de usuario.

7.3.1 Capa *Model*

La base de datos *Firebase* es un contenedor vacío, sin lógica ni modelo, que ha de ser modelado enteramente por la aplicación que accede a él. Por lo tanto, toda la implementación es parte de la aplicación. Para asegurar que las referencias y la implementación se mantenga según lo diseñado y que los accesos son coherentes, en el modelo se crearon enumerados con los nombres de las columnas de cada colección como podemos ver en el ejemplo de la figura 7.3.

En esta capa también tenemos el *repositorio*, uno por cada colección, que son los encargados de realizar las consultas a la base de datos. Todos los repositorios hacen una carga *by lazy* de las referencias a la colección correspondiente. Esto quiere decir que hasta no ser nece-

```
private val valoracionDb by lazy {
    FirebaseFirestore.getInstance().collection("Valoracion")
}

fun borrarValoracion(id: String): Task<Void>{
    Log.d(TAG, "borrarValoracion()")
    return valoracionDb.document(id).delete()
}
```

Figura 7.4: Ejemplo de inicialización *by lazy* de instancia a ***Valoracion*** y un borrado a partir de un id usando esta instancia. ***Localizacion***

```
class LoginViewModelFactory(private val repository: AuthRepository)
    : ViewModelProvider.NewInstanceFactory() {
        override fun <T : ViewModel?> create(modelClass: Class<T>)
            : T {
                return LoginViewModel(repository) as T
            }
}
```

Figura 7.5: Ejemplo de creación de una factoría.

sario utilizarlas, no se inicializan. A partir de estas instancias, se realizan todas las consultas correspondientes.

En la figura 7.4 podemos ver un ejemplo de una carga *by lazy* de la colección *Valoracion*, y como a partir de ella se realiza el borrado de una *valoracion* a partir de su id.

7.3.2 Capa *View-Model*

Esta capa es la intermediaria entre las *views* que necesitan información de la base de datos y el *Model*. Por ello es importante que exista un *View-Model* por cada *View* que necesita datos de la base de datos.

Para facilitar la *escalabilidad* y *flexibilidad* del proyecto, se ha hecho uso del concepto de *inyección de dependencias* a la hora de conectar el *View-Model* con el *Model*. Asegurándonos de esta manera un acoplamiento débil, es decir, una interdependencia mucho menor.

Para llevar a cabo esto, se hizo uso de *Kodein*, de tal manera que que cada elemento del *view-model* hace uso de una instancia de cada repositorio, y no crea uno propio.

Para su implementación, se crear una *factoría* que de lo único que se encarga es de crear el *View-Model* a partir de unas instancias del repositorio. Podemos ver un ejemplo de esto en la figura 7.5

Una vez creada la factoría, se declara en una clase que gestiona las dependencias que extiende de *Application()* y *KodeinAware*. En este archivo se declaran los repositorios como un

```
class ApplicationDependencies : Application(), KodeinAware {
    override val kodein = Kodein.lazy {
        import(androidXModule(this@ApplicationDependencies))

        bind() from singleton { AuthRepository() }
        bind() from provider { LoginViewModelFactory(instance()) }
    }
}
```

Figura 7.6: Ejemplo de declaración de dependencias para el ejemplo de la figura 7.5.

```
    override val kodein by kodein()
    private val loginViewModelFactory: LoginViewModelFactory by
    instance()
    private lateinit var  loginViewModel : LoginViewModel

    ...

    private fun initLoginViewModel(){
        loginViewModel = ViewModelProviders.of(this,
    loginViewModelFactory).get(LoginViewModel::class.java)
    }
```

Figura 7.7: Inicialización del *ViewModel* desde la *View*.

Singleton, y las factorías como *providers* que recibirán instancias en su constructor. También se inicializa una variable llamada *kodein*, que llevará incluidas todas estas dependencias.

En la figura 7.6 se puede ver cómo se realiza todo este proceso. Por simplicidad, se emplean las mismas referencias a los repositorios y factorías usadas en el ejemplo de la figura 7.5.

En este punto es importante declarar esta clase, en este caso *ApplicationDependencies*, en el apartado `<application>` del *Manifest* en el `android:name`. De esta forma, cuando se inicie el proceso de la aplicación, se crean las instancias de esta clase antes que de los componentes de la aplicación. Una vez realizados todos estos pasos, lo único que quedaría por llevar a cabo es hacer la llamada desde la *view*, que como vemos en el ejemplo de la figura 7.7, no tiene por qué saber nada de la implementación del repositorio.

7.3.3 Capa *View*

En lo relativo a esta capa, la característica a resaltar es que se buscó en todo momento seguir manteniéndola ajena a la implementación del *Model*. Por ello, siempre que era necesario acceder a valores o editarlos, sólo podía hacerlo a través del *ViewModel*. Un ejemplo lo podemos ver en la figura 7.8 en la que se asigna un valor a un *TextView* a partir de los datos en el *ViewModel*.

```
1 location_name_tv.text = mapViewModel.selectedLocation!!.nombre
```

Figura 7.8: Ejemplo de acceso a un dato desde la *View* usando el *ViewModel*.

Figura 7.9: Logo *PopAdvisor*

7.3.4 Interfaz de usuario

Aunque formalmente este apartado debería estar dentro de la *View*, hemos preferido dedicarle una sección proppia para centrarse más en el apartado visual a la hora de establecer un manejo intuitivo para el usuario.

La interfaz se montó a partir del logo, figura 7.9, que fue lo primero que se creó, y con él se establecieron las bases de los detalles del diseño gráfico. Los colores que aparecen en el logo establecieron la jerarquía visual del resto de elementos gráficos:

- Azul: El color primario de la aplicación y que genera toda la gama de azules usados para presentar la información.
- Rojo: Color secundario presente en todo lo referente a las acciones y componentes con los que el usuario puede interactuar.
- Blanco: Color neutro que sirve de base donde se fundamenta el resto de colores para no sobrecargar.

Además de los colores, las formas de los componentes visuales se adaptaron a las formas propuestas por el logo, generando así botones con acabados redondeados, iconos simples sin demasiado detalle y buscando la idea en todo momento de evitar generar pantallas y componentes con demasiadas características, buscando mantener al mínimo el detalle. En la figura 7.10 podemos ver algún ejemplo de lo que se comenta en este apartado.

Figura 7.10: Captura de una de las pantallas de la aplicación Android.

Capítulo 8

Planificación y costes

En este capítulo se detallan en primer lugar los distintos recursos disponibles y a continuación se plantea la planificación inicial prevista a partir de las historias de usuario del proyecto y su descomposición en *sprints*. También se valoran los riesgos previstos inicialmente y los planes de contingencia contamplados para cada uno de ellos. Una vez detalladas estas cuestiones, se describe el seguimiento del proyecto analizando los resultados de cada *sprint* y modificando la planificación en función de los retrasos surgidos. Se cierra el capítulo con un análisis de los costes.

8.1 Recursos

Los recursos disponibles se han divido en recursos humanos, recursos materiales y recursos software.

8.1.1 Recursos humanos

Los recursos humanos disponibles para este proyecto se dividen en tres:

- **Jefe de proyecto:** Director del proyecto que realiza las tareas de supervisión, analista del desarrollo y el que toma las decisiones de diseño. Este papel estará desempeñado por el alumno.
- **Analista desarrollador:** El que implementa y desarrolla el proyecto. También se hará cargo el alumno.
- **Asesor:** Persona de confianza y con experiencia que aconsejará y guiará en distintas cuestiones del desarrollo que puedan surgir, así como de la documentación. Papel desempeñado por el tutor del proyecto.

8.1.2 Recursos materiales

Los recursos materiales fueron principalmente dos.

- Ordenador portátil con Windows 10 con el que se llevó a cabo todo el desarrollo, diseño e implementación.
- Dispositivo móvil del alumno empleado para probar la aplicación durante su desarrollo. Un Xiaomi Mi 8 Lite con Android 9.0. La aplicación final también se ha probado con otros dispositivos móviles.

8.1.3 Recursos software

A continuación se presentan los recursos software empleados durante el desarrollo.

- Para el desarrollo Android, se ha hecho uso de Android Studio, la herramienta gratuita ofrecida por Google.
- Para la gestión del almacenamiento y base de datos de Firebase, se hizo uso de las herramientas online que ofrece *FirebaseConsole* para la gestión.
- Para el diseño de la aplicación, se hizo uso de *Figma*, una herramienta gratuita de diseño de interfaces de usuario.
- Para el diseño de logos e iconos personalizados, se hizo uso de *Inkscape*, una herramienta de dibujo vectorial de código libre.
- Para el diseño de animaciones personalizadas se hizo uso de *After Effects*.

8.2 Planificación inicial

Según la metodología XP seleccionada, para explicar la planificación inicial prevista, empezaremos con las historias de usuario, y seguiremos con los *Sprints* y las *Release Dates*. Sobre esta planificación, se puede ver una representación gráfica en el diagrama de Gantt de la figura 8.1.

8.2.1 Historias de usuario

A continuación se detallan las historias de usuario especificadas al principio del proyecto y las horas estimadas en este punto, horas en las cuales se incluye el tiempo de diseño de interfaz, desarrollo, pruebas posteriores y documentación.

- **Planificación inicial:** Elaboración de las historias de usuario, investigación de las arquitecturas y de las distintas tecnologías disponibles, planificación de los *sprints* y *release dates*.
  Estimado: 15 horas.

- **Montar estructura inicial:** Crear un proyecto nuevo con la estructura inicial basada en la arquitectura MVVM explicada en el apartado 6.1.4, crear repositorio remoto y conectar Firebase con la aplicación.
  Estimado: 15 horas.

- **Implementar autenticación por Google:** Investigar cómo implementar y adaptar a la arquitectura, y desarrollar la funcionalidad para iniciar sesión a partir de las credenciales de Google. Añadir recordatorio para quien no lo haga que inicie sesión.
  Estimado: 15 horas.

- **Cargar localizaciones ceracanas:** Llevar a cabo primero una pequeña prueba de concepto trayendo todos los datos de la misma ciudad, para luego investigar cómo llevar a cabo el sólo traer las localizaciones en las cercanías. Crear datos de prueba en la base de datos para probar.
  Estimado: 25 horas.

- **Crear localización:** Crear localización. Sólo lo podrán hacer los usuarios autenticados, y sólo se pedirá nombre y descripción de la localización.
  Estimado: 15 horas.

- **Cargar localización:** Al seleccionar una localización se cargarán los datos asociados a ella.
  Estimado: 15 horas.

- **Cómo llegar:** Añadir en las localizaciones la funcionalidad de cómo llegar.
  Estimado: 5 horas.

- **Editar localización:** Funcionalidad para editar el nombre y/o la descripción. Sólo dejar a los usuarios autenticados realizarla y sólo a aquel que la haya creado.
  Estimado: 7 horas.

- **Borrar localización:** Funcionalidad para borrar una localización. Sólo dejar a los usuarios autenticados realizarla y sólo a aquel que la haya creado.
  Estimado: 7 horas.

- **Crear valoración:** Funcionalidad para crear una valoración. Sólo la podrán llevar a cabo los usuarios autenticados.
  Estimado: 15 horas.

- **Cargar valoraciones:** Cargar las valoraciones asociadas a una localización.
  Estimado: 15 horas.

- **Editar valoración:** Funcionalidad para editar el comentario de una valoración. Sólo dejar a los usuarios autenticados realizarla y sólo a aquel que la haya creado.
  Estimado: 5 horas.

- **Borrar valoración:** Funcionalidad para borrar una valoración. Sólo dejar a los usuarios autenticados realizarla y sólo a aquel que la haya creado.
  Estimado: 5 horas.

- **Cargar imágenes:** Cargar las imágenes asociadas a una valoración. Crear imágenes de prueba en base de datos.
  Estimado: 15 horas.

- **Subir imagen:** Funcionalidad para subir una imagen. Sólo la podrán llevar a cabo los usuarios autenticados.
  Estimado: 7 horas.

- **Subida y despliegue a playstore:** Investigar y llevar a cabo la publicación en playstore.
  Estimado: 7 horas.

- **Elaboración final documentación:** Documentar el proyecto..
  Estimado: Se ha decidido dejar sin estimar porque dependerá principalmente de la cantidad de documteación generada hasta este punto.

8.2.2 Sprints

Los sprints, dado que el alumno ha tenido que compaginar la elaboración del proyecto con una jornada laboral, están compuestos de 30 horas de trabajo desarrolladas a lo largo de dos semanas, aunque se aprovechará la flexibilidad de la metodología para en algún *sprint* no siempre ser tan rígidos con esas 30 horas. Al estar trabajando con *Extremme Programming*, la siguiente enumeración de los *sprints* y lo que se llevará a cabo en cada uno, es una guía que puede ir cambiando y evolucionando a medida que se van desarrollando.

- **Sprint 1**

1. Planificación inicial
2. Montar estructura inicial

- **Sprint 2**

1. Implementar autenticación por Google
2. Cargar todas las localizaciones de la misma ciudad

- **Sprint 3**

1. Cargar localización
2. Cómo llegar
3. Crear localización

- **Sprint 4**

1. Editar Localización
2. Borrar Localización
3. Crear Valoración

- **Sprint 5**

1. Cargar Valoraciones
2. Editar Valoración
3. Borrar Valoración

- **Sprint 6**

1. Cargar Imágenes
2. Subir Imagen

- **Sprint 7**

1. Subida y despliegue a Playstore
2. Elaboración final documentación

8.2.3 Plan de entregas (*Release Dates*)

Las *release dates* que se han decidido marcar, con sus funcionalidades correspondientes, son:

1. **Final Sprint 2:** Se cargan los datos creados en base de datos de forma manual de la misma ciudad, y se puede acceder a la aplicación mediante las credenciales de Google.

2. **Final Sprint 4:** Se cargan las localizaciones en un radio de 500 metros, se puede ver la información asociada a cada una, saber cómo llegar y también crear localizaciones.

3. **Final Sprint 6:** Se pueden editar y borrar localizaciones, como también crear, cargar, editar y borrar valoraciones.

4. **Final:** Se puede acceder a la aplicación desde la Playstore con todas las funcionalidades, incluidas cargar y subir imágenes.

8.3 Riesgos y planes de contingencia

En este apartado analizaremos los riesgos ponedreando su probabilidad de ocurrencia e impacto en el proyecto, para luego realizar planes de contigencia.

8.3.1 Riesgos

A continuación tenemos los riesgos contemplados, y en la tabla 8.1, la ponderación de su probabilidad de ocurrencia y del impacto estimado en el proyecto.

- **Formación en Kotlin:** El equipo de desarrollo no tiene formación ni experiencia anterior en este lenguaje de programación.

- **Uso de herramientas externas desconocidas:** El equipo de desarrollo no tiene experiencia trabajando con *Firestore*, *Cloud Storage*, ni con los servicios de autenticación de *Firebase*.

- **Alta complejidad al trabajar con imágenes:** El trabajar con subida y cargas de imágenes puede complicar la gestión de memoria de la aplicación Android.

- **Dedicación de tiempo del equipo de desarrollo:** El equipo de desarrollo está constituido únicamente por el alumno, tal y como ya se comentó anteormente. Además se debe tener en cuenta que durante la realización del proyecto estará trabajando y deberá asistir a clases y realizar prácticas y exámenes, por lo que en algunas semanas habrá una dedicación menor a la planificada.

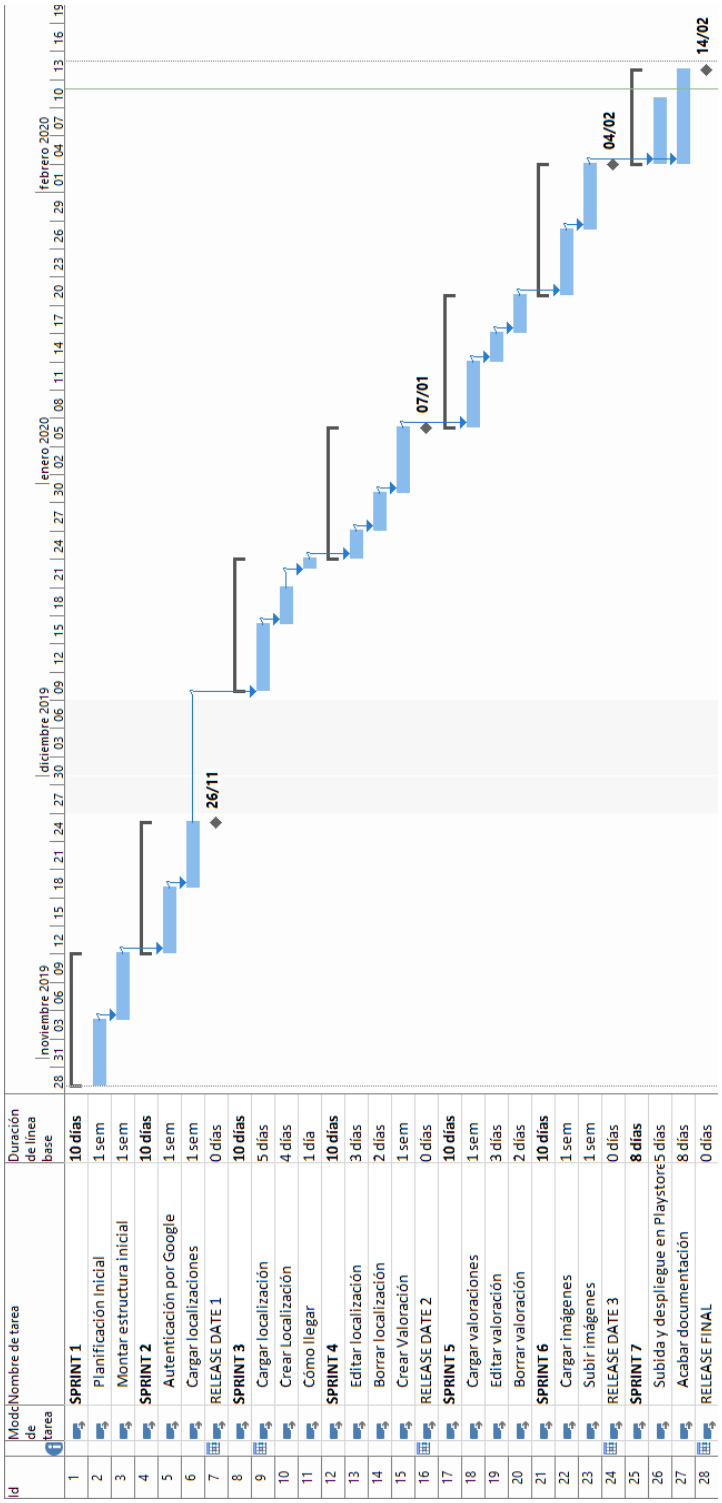


Figura 8.1: Diagrama de Gantt de la planificación inicial del proyecto

Tabla 8.1: Riesgos previstos para el proyecto, junto con su probabilidad e impacto estimados.

Riesgo	Probabilidad	Impacto
Formación en Kotlin	Alta	Medio
Uso de herramientas externas desconocidas	Alta	Medio
Alta complejidad al trabajar con imágenes	Media	Alto
Dedicación de tiempo del equipo de desarrollo	Alta	Media

8.3.2 Planes de contingencia

En este apartado trataremos las formas de prevenir cada uno de los riesgos explicados y evaluados anteriormente.

- **Formación en Kotlin:** Los primeros *sprints* formarán parte de la adaptación al nuevo lenguaje, por ello se han estimado unas horas de una forma holgada para afrontar este apartado. En caso de que la curva de aprendizaje sea un poco más compleja de lo esperado, la flexibilidad de la metodología facilitará la replanificación y readaptación del proyecto.
- **Uso de herramientas externas desconocidas:** El plan de contingencia para este caso seguirá los mismos pasos que el anterior.
- **Alta complejidad al trabajar con imágenes:** Al tratarse de un posible impacto *alto*, se decide dejar como último a implementar.
- **Dedicación de tiempo del equipo de desarrollo:** Se ha buscado dotar a los sprints de un número de horas no demasiado *optimista*, y también hay ciertas semanas, que como se puede ver en el diagrama de Gantt, están planificados ciertos parones. Esto dejará cierto margen para recuperar algo de tiempo.

8.4 Seguimiento planificación

A continuación se comentan los aspectos más relevantes sobre el seguimiento del desarrollo del proyecto a lo largo de los distintos sprints. Sobre este seguimiento, se puede ver una representación gráfica en el diagrama de Gantt de la figura 8.2 de la replanificación y adaptación que se tuvo que dar en el desarrollo de las iteraciones.

- **Sprint 1:** El desarrollo de este *sprint* se realizó según lo planificado.
- **Sprint 2:** A la hora de implementar la autenticación surgieron varios problemas relacionados con la curva de aprendizaje y la complejidad añadida de implementarla correctamente en la arquitectura MVVM. Se realizó una primera versión sin seguir la arquitectura, para luego adaptarla a ella.

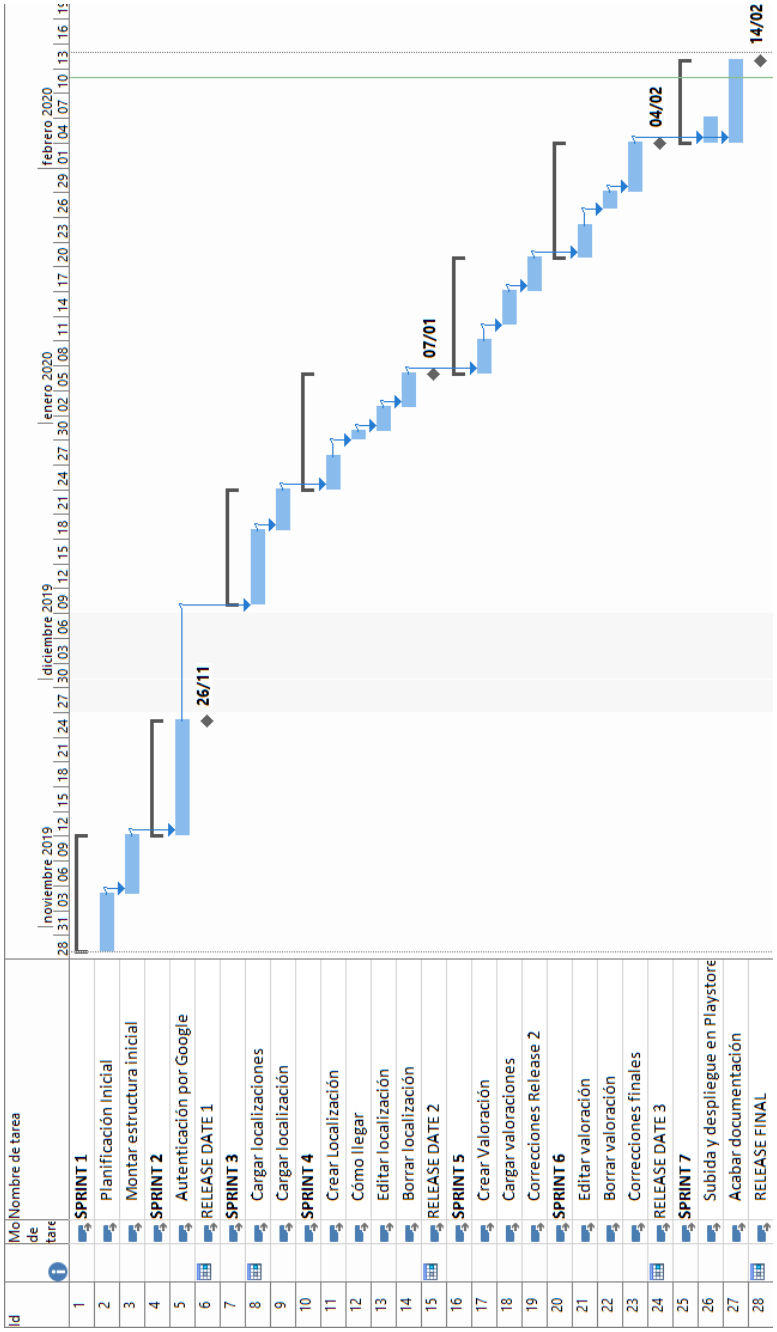


Figura 8.2: Diagrama de Gantt del seguimiento del proyecto.

Además, *Cargar localizaciones cercanas* se movió al siguiente *sprint*, por lo que se tuvo que replanificar. Se llega a la primer *Release date* sólo con la autenticación por Google.

- **Sprint 3:** Aunque la curva de aprendizaje entre *Kotlin* y las herramientas de *Firebase* se fue suavizando tras el *sprint* anterior, el parón por exámenes tuvo un impacto mayor del previsto.

  Como respuesta se decidió eliminar de este *sprint* las funcionalidades de *crear localización* y *cómo llegar*, y replanificar otra vez los siguientes *sprints*.

  Comprobando lo sucedido hasta este punto, se decidió hacer una previsión aún más *realista* y se decidió descartar las funcionalidades relacionadas con imágenes, y mantenerlas en el caso excepcional de que sobrase tiempo.

- **Sprint 4:** Tras la nueva planificación, los objetivos de este *sprint* eran implementar las funcionalidades de *crear localización*, *cómo llegar*, *editar localización* y *borrar localización*, que se consiguió sin mayor problema.

  Se llega a la segunda *Release date* con un *sprint* acorde a las funcionalidades esperadas tras las replanificaciones, pero no con respecto a la planificación inicial.

  También se encuentran los siguientes detalles y errores a corregir:

  - Falta control ante errores.
  - Controlar en qué idioma se guardan los nombres de las ciudadas y países, y también se muestran.
  - Agregar info del usuario logueado en alguna parte.
  - Retocar maquetación vistas *Localización*.
  - Bug: La Valoración media de las localizaciones no se está cargando bien.

- **Sprint 5:** Este *sprint* se desarrolló según lo esperado tras la *Release date*. Las funcionalidades *crear valoración* y *cargar valoraciones* quedaron implementadas, dejando *editar valoración* y *borrar valoración* para el último *sprint*.

  En este *sprint* también se sacó una pequeña lista de detalles y pequeños errores a corregir para el siguiente (principalmente por ser el último y no tener previsto una gran carga de trabajo al haber descartado las funcionalidades de imágenes):

  - La funcionalidad de *cargar localización* en la vista, tiene un problema de lógica de funcionamiento en las transiciones entre *OnPause()* y *OnResume()*.
  - *Crear localización* no es intuitivo, investigar e implementar una forma mejor.
  - Se está omitiendo si el usuario logueado es un usuario *Administrador*.

Tabla 8.2: Costes de recursos humanos del proyecto.

Recurso	Salario (€/h)	Trabajo (h)	Coste (€)
Jefe de proyecto	50	33	1.650
Desarrollador	25	180	4.500
Diseñador UI	30	17	510
Consultor	43	33	1.419
Asesor	40	28	1.120
Total			7.929

- **Sprint 6:** Este *sprint* se desarrolló según lo replanteado a lo largo de las iteraciones.
- **Sprint 7:** Este *sprint* se desarrolló según lo planteado en la planificación inicial.

8.5 Costes

A partir de la planificación inicial y los recursos humanos y materiales empleados, en los siguientes apartados se detallan los costes del proyecto.

8.5.1 Costes de recursos humanos

Para realizar un estimación de los costes humanos se ha estimado su cantidad de horas de trabajo en el proyecto y su coste por hora.

- Jefe de proyecto: El primer *sprint* cinco días y una jornada de tres horas, y tres horas al finalizar cada uno de los siguientes *sprints* para reuniones de seguimiento.
- Desarrollador: A partir del segundo *sprint* Cinco días a la semana y una jornada de tres horas.
- Diseñador UI: En el primer *sprint* cinco horas, y en los siguientes dos horas al principio de cada uno.
- Consultor: El primer *sprint* cinco días y una jornada de tres horas, y en lo siguientes tres horas al finalizar para reuniones.
- Asesor: Cuatro horas por cada sprint para consultas y reuniones en algunos casos específicos.

El desglose en cuanto a costes totales lo podemos ver en la tabla 8.2.

Tabla 8.3: Costes de materiales del proyecto.

Recurso	Coste	Imputado al proyecto
Ordenador portátil (Asus)	599€	0€
Teléfono móvil (Xiaomi Mi 8 Lite)	189€	0€

Tabla 8.4: Costes de la herramienta Firebase según los planes *Spark* y *Blaze*.

	Spark	Blaze
Firestore		
Datos almacenados	1 GiB en total	USD 0.18 por GiB
Operaciones de escritura	20,000/día	USD 0.18 por cada 100,000
Operaciones de lectura	50,000/día	USD 0.06 por cada 100,000
Operaciones de eliminación	20,000/día	USD 0.02 por cada 100,000
Storage		
GB almacenados	5 GB	USD 0.026/GB
GB descargados	1 GB/día	USD 0.12/GB
Operaciones de carga	20,000/día	USD 0.05 por cada 10,000
Operaciones de descarga	50,000/día	USD 0.004 por cada 10,000
TestLab		
Pruebas en dispositivos virtuales	10 pruebas por día	USD 1 por dispositivo por hora
Pruebas en dispositivos físicos	5 pruebas por día	USD 5 por dispositivo por hora

8.5.2 Costes materiales y software

En cuanto a recursos materiales, al haber hecho uso de recursos a los que ya se tenía acceso, no se han sumado al coste. Podemos ver el desglose en la tabla 8.3

En el caso del software, sólo se ha utilizado software de acceso gratuito, licencias académicas o pruebas gratuitas, por lo que no se imputa ningún coste.

Es importante indicar que Firebase ofrece su servicio en dos planes. Para el desarrollo de nuestro proyecto hemos empleado el plan *Spark* que es gratuito y ofrece todas las funcionalidades que necesitamos. Sin embargo, de cara a la escalabilidad de nuestra aplicación, hay que tener en cuenta las limitaciones del plan empleado y los costes del plan *Blaze*, comparación que podemos ver en la tabla 8.4 [38].

8.5.3 Costes finales

En cuanto a los *recursos humanso*, aunque hubo un retraso en el desarrolllo de los *sprints*, al tomar la decisión de descartar funcionalidades para mantener los tiempos, no conllevó un aumento de horas de trabajo, lo que no desencadenó en un sobrecoste sobre lo planificado. En lo relativo a los *recursos materiales y software*, lo único que tenía cierto nivel de incertidumbre era el uso de *versiones de prueba* y no excederse en las interacciones con *Firebase* teniendo que cambiar de plan *Spark* a plan *Blaze*, por lo que al no haberse excedido ni en una cuestión ni en otra, tampoco este apartado generó ningún sobrecoste, lo que sumado a los resultados en

costes de *recursos humanos*, nos deja que el proyecto se desarrolló acorde lo estimado.

Capítulo 9

Pruebas

En este capítulo trataremos las distintas pruebas que se llevaron a cabo: Pruebas a final de *sprint*, las herramientas de monitorización utilizadas, y el cuestionario de calidad que se generó para usuarios reales con los que se hizo una prueba al final del desarrollo.

9.1 Pruebas de *caja negra*

Como se comentaba en el apartado 5.2, cada *sprint* tiene un tiempo reservado para pruebas. Se realizaron dos tipos de pruebas: al final de cada *sprint* y en las *releases*.

9.1.1 Pruebas al final de cada *sprint*

Estas pruebas buscaron en todo momento probar las funcionalidades desarrolladas en el *sprint*. Se buscaba comprobar el correcto funcionamiento de las funcionalidades desarrolladas, utilizando como guía el documento generado al principio de cada uno que detallaba cómo debía funcionar y qué restriccions debía cumplir. Los resultados que se iban obteniendo se analizaban desde la interfaz de la aplicación, que no se cerrara la aplicación de forma inesperada, o que no surgiera nada inesperado, y desde la base datos utilizando la consola de *Firebase*, comprobando que los datos creados, borrados y/o editados eran correctos y tenían coherencia con la última acción o con lo que se muestra en la interfaz. Al dejar tiempo para las pruebas en el propio *sprint*, todo *bug* encontrado en él, se intentaba solucionar en él, si no era posible, se creaba una lista de *bugs* para solucionar en el siguiente *sprint*.

9.1.2 Pruebas *releases*

Estas pruebas son muy parecidas a las de final de *sprint*, pero estas, se centraban principalmente en la correcta integración de las funcionalidades desarrolladas hasta el momento. Otra de las diferencias con las anteriores, es que estas pruebas no sólo generaban un lista de *bugs*, sino que a su vez generaban una lista de *detalles a mejorar*, que consistían en cuestiones

de la aplicación que no eran un *bug* propiamente dicho, pero que por distintas razones era una buena idea cambiar o añadir. Esta lista se dividía en tres principalmente:

- Funcionamiento IU: Enfocadas en cualquier funcionamiento que no fuese del todo intuitivo o que se pudiese mejorar.
- Fidelidad con el diseño inicial: Enfocadas en cualquier parte del código que no estuviese siendo fiel con el diseño inicial.
- Rendimiento: Enfocadas en cualquier acción que pudiese estar tardando más de lo esperado, a pesar de estar funcionando correctamente.

Esta idea de la lista de *detalles a mejorar* nace de la práctica de *Recodificación* de *Extremme Programming*, explicada en el apartado 5.2.2.

9.2 Monitorización

A mayores de las pruebas de *caja negra* que involucraban al equipo de desarrollo probando de forma directa, si hizo uso de distintas herramientas que o probaban la aplicación de forma automática y nos ofrecía un informe detallado de la ejecución, o en lugar de probarla, monitorizaban y aportaban detalles y estadísticas sobre la ejecución de la aplicación, por el equipo de desarrollo o por usuarios reales.

9.2.1 *Firebase Analytics*

A la hora de generar un aplicación estable para dar a usuarios *testers*, se comenzó a usar esta herramienta, que de lo que se encarga principalmente es de ayudar a comprender cómo las personas usan tu app. El SDK captura automáticamente diversos eventos y propiedades del usuario, y también permite definir propios eventos personalizados a fin de medir factores particularmente importantes. Una vez capturados los datos, se pueden visualizar en un panel mediante Firebase console. Este panel proporciona estadísticas detalladas sobre los datos; desde datos de resumen, como los usuarios activos y los segmentos demográficos a los que pertenecen, hasta información más detallada, como los artículos más comprados.

Tras una sucesión de pruebas con esta herramienta, se llegó a la conclusión, de que a pesar que sí que ayudaba a comprender cómo interactuaban los usuarios con la aplicación, al estar en medio del desarrollo, los valores que se muestran no son representativos al no tener un público real, constante y establecido, como podemos ver en la figura 9.1 que nos muestra el porcentaje del tiempo que pasan los usarios en cada activity donde la media de tiempo más alta está en cuatro segundos, lo que también genera que según qué días, los eventos por sesión fluctúan demasiado, como podemos ver en la figura 9.2.

Interacción de los usuarios > Clase de pantalla

Clase de pantalla	% del total		Tiempo medio	
MapActivity	48,92 %	↓ 36,8 %	0 mi...4 s	↓ 61,3 %
HomeActivity	16,81 %	↓ 21,3 %	0 mi...5 s	↓ 20,4 %
ReviewActivity	10,39 %	-	0 mi...5 s	-
LocationActivity	8,39 %	-	0 mi...0 s	-
SearchActivity	6,92 %	-	0 mi...4 s	-
LoginActivity	4,61 %	↑ 47... %	0 mi...1 s	↓ 25,7 %
IntroActivity	1,89 %	↑ 30... %	0 mi...5 s	↓ 20,1 %
ReviewListActivity	1,85 %	-	0 mi...1 s	-

Figura 9.1: Ejemplo de informe del porcentaje de uso de cada *Activty* en *Analytics*.

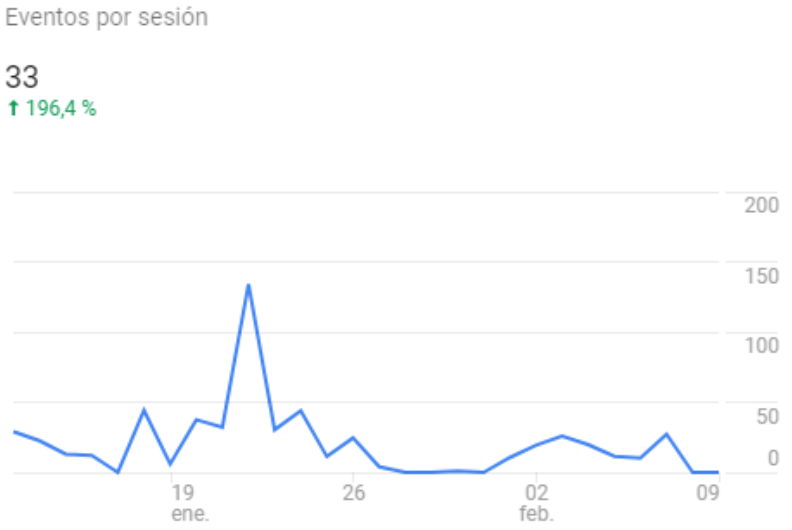


Figura 9.2: Ejemplo de informe de eventos por sesión en *Analytics*.

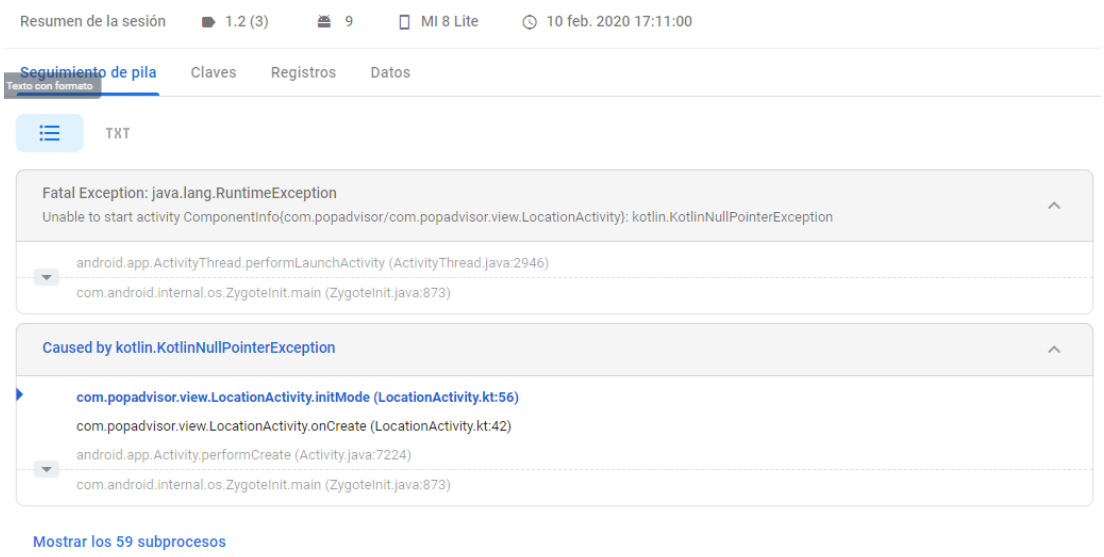


Figura 9.3: Ejemplo generado de forma intencionada para mostrar el informe generado por *Crashlytics*.

9.2.2 Firebase *Crashlytics*

Crashlytics se trata de otra herramienta hecha para generar informes sobre el rendimiento de la aplicación, pero en este caso enfocada en informar fallas en tiempo real, hacer un seguimiento de los problemas de estabilidad que afectan la calidad de la app y priorizarlos. Agrupa las fallas de forma inteligente y destaca las circunstancias en las que se produjeron, lo que permite ahorrar tiempo en la solución de problemas. A diferencia de *analyticis*, no es una herramienta que venga por defecto integrada a nuestra aplicación, sino que hay que seguir una serie de pasos muy sencillos para hacerlo que vienen muy explicados en la guía de *Firebase* [39].

Una vez integrado, sólo es cuestión de ejecutar la aplicación, y ante cada fallo o error, la consola de *Firebase* para *Crashlytics*, nos mostrará un informe del error ocurrido, con detalles del dipositivo, el mensaje de error, la fecha y la hora, y demás detalles que puedan ser de utilidad a la hora de corregirlo, como podemos ver en el ejemplo de la figura 9.3,

9.2.3 Firebase *Performance*

Firebase Performance Monitoring es un servicio que permite obtener estadísticas sobre las características de rendimiento la aplicación. Recopila datos de rendimiento, y los facilita en la *Firebase console*. Ayuda a comprender dónde y cuándo se puede mejorar el rendimiento. Al igual que *Crashlytics*, no es una herramienta que venga por defecto integrada en nuestra aplicación, sino que hay que seguir una serie de pasos muy sencillos para hacerlo que vienen

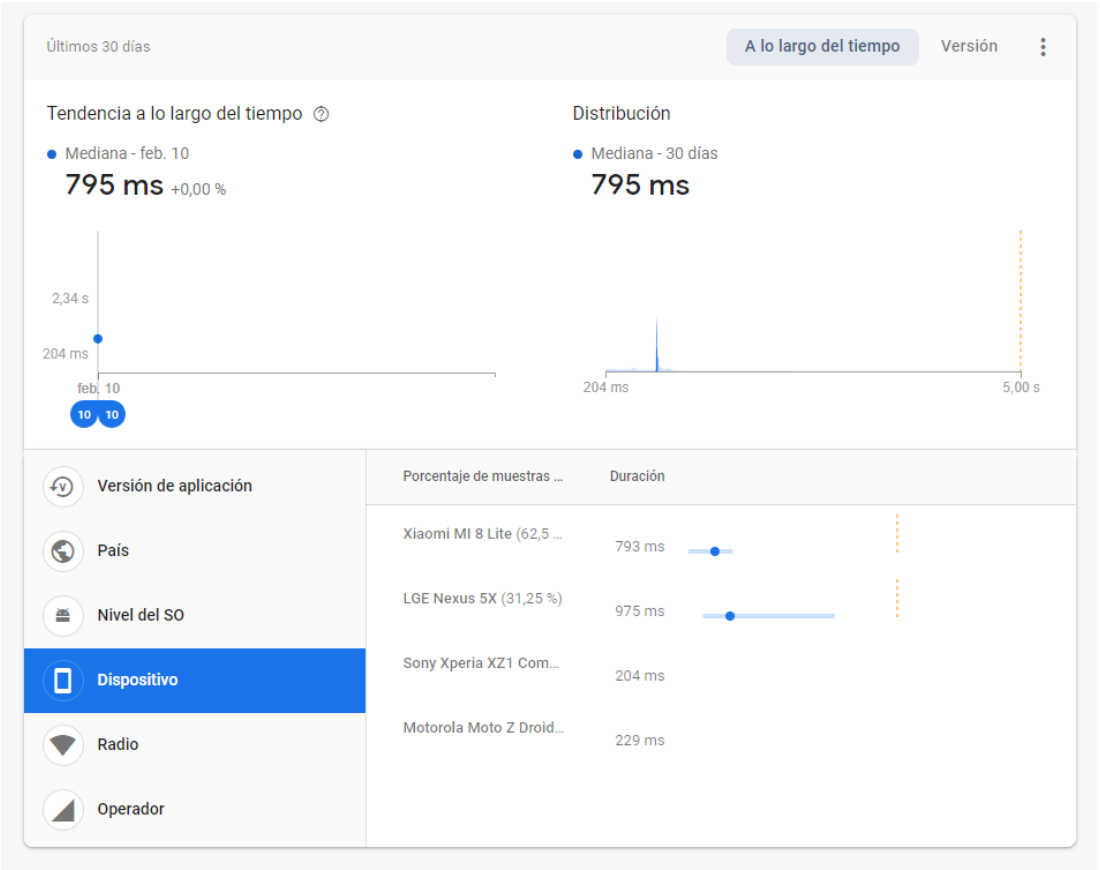


Figura 9.4: Ejemplo de estadísticas de arranque ofrecidas por *Firebase Performance* sobre los distintos dispositivos en los que ha sido ejecutada la aplicación.

muy bien explicados en la guía de *Firebase* [40].

De entre las distintas posibilidades que nos ofrece, una de las más interesantes es la información sobre cuánto tiempo suele tardar en arrancar la aplicación, ofreciéndonos esta información con una mediana del tiempo, y diviéndolo entre varios paráemtros como pueden ser modelo de móvil, versión de la aplicación, operador, o nivel de sistema operativo de *Android*. Es una forma muy fácil de detectar si estamos teniendo problemas de rendimiento de nuestra aplicación. Al ofrecernos informes de la ejecución en distintos dispositivos, o versiones de *Android*, facilita unas posiblidades que el desarrollador de forma física no podría conseguir. En la figura 9.4 tenemos un pequeño ejemplo sobre estas estadísticas.

9.2.4 Firebase *TestLab*

Testlab es una herramienta que ofrece la consola de *Firebase* para poder probar aplicaciones de forma automatizada [41]. Dentro de las opciones que se ofrecen, se hizo uso de *Robo*, que lo que hace es a partir de un *apk*, despliega y ejecuta la aplicación en los dispotivios que

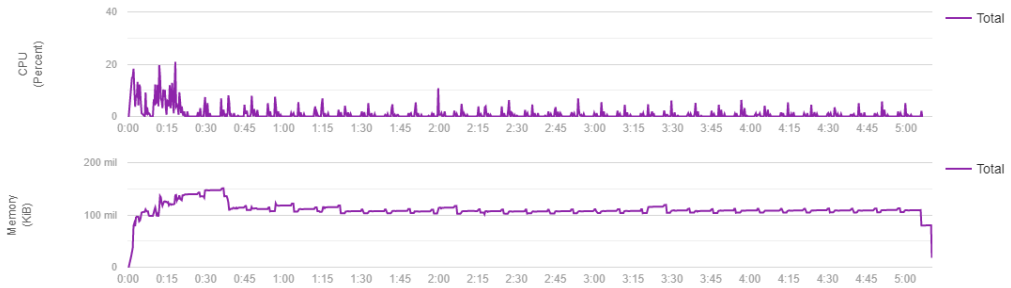


Figura 9.5: Ejemplo de estadísticas sobre el uso de la CPU y de la memoria ofrecidas por *TestLab*.

se le indiquen y realiza una serie de acciones completamente aleatorias con la intención de provocar situaciones que probablemente el desarrollador no llegase a preveer para probar su funcionamiento.

Una vez realizadas las pruebas, se genera un resumen de las ejeuciones, con estadísticas, capturas y demás información extraída de la prueba, que pueden ser muy útiles para el desarrollador. En la figura 9.5 tenemos un ejemplo de las estadísticas sobre uso de memoria y CPU de la ejecución de una aplicación durante cinco minutos en un Samsung Galaxy S9 SM-G960F, nivel de API 26.

Otra gran ventaja es la gran variedad de dispositivos con los que se pueden realizar pruebas, algo inalcanzable para un equipo de desarrollo de tan pequeña magnitud.

9.2.5 Google Play

Una vez finalizados los sprints que se enfocaban en desarrollar las funcionalidades de la aplicación, se lanzó una beta abierta en Google Play. Las betas abiertas se diferencian de las cerradas en que el administrador de la beta no define una lista de cuentas que puedan acceder, sino que genera un enlace mediante el que se accede a ella. Esto se hizo con la intención de que se compartiera el enlace y alcanzar un mayor número de usuarios.

La beta de Google Play, además de recopilar datos de instalación y uso, permite al usuario reportar cada error de la aplicación incluyendo un comentario que es recibido en el correo electrónico del desarrollador.

9.3 Cuestionario experiencia de usuario

Durante la etapa final, se distribuyó la *beta* publicada en la *PlayStore* a un grupo reducido de personas para poder realizar pruebas ajenas a la automatización y al desarrollador. Para evaluar la experiencia de los distintos *testers* se creó un formulario para recopilar la información considerada relevante. Los resultados los encontramos en la tabla 9.1

Tabla 9.1: Resultados del cuestionario sobre la experiencia de usuario con la aplicación beta del Google Play.

Pregunta	**Nota**
Aplicación intuitva	4.44/5
Facíl de aprender a manejar	4.67/5
Facíl de usar	3/5
Aplicación atractiva conceptual y visualmente	4/5
Se encuentra fácil lo que se busca	4.55/5
La información que se muestra se entiende fácil	4.77/5
Aplicación útil	3.77
Recomendable a amigos	4.75
Aplicación con funcionalidades únicas	4.55
Cierres inesperados de la aplicación	11.1%
Tarda en cargar	1.66/5
Sucesos inpeserados en la ejecución	22.2%

Como podemos observar en los resultados, podríamos concluir que a este pqeueño grupo de *testers* les ha parecido una aplicación muy intuitiva, que no cuesta aprender a manejarla, pero que al mismo tiempo, no es tan fácil de usar. A nivel organizativo y visual, podemos ver, según los resultados, es una aplicación atractiva, que llama la atención, que no es difícil encontrar lo que se busca, y que la información que aporta es fácil de comprender. En cuanto a usabilidad, se comprueba sobre estos resultados que es una aplicación que se podría beneficiar del boca a boca por el alto resultado en si se recomendaría a amigos y tiene funcionalidades únicas, pero al mismo tiempo nos encontramos con que no parece a priori una aplicación tan útil. A nivel rendimiento, aunque los resultados no son malos propiamente dicho, el que el 11'1% de los usuarios haya tenido cierres inesperados de la aplicación y un 22'2% sucesos inesperados en la ejecución, sugiere que quedan varios *bugs* por solventar. Concluir que aunque en general es un buen resultado sobre la experiencia de los usuarios, es un grupo pequeño de *testers* y la gran mayoría, gente cercana al equipo de desarrollo, por lo que no se puede tomar al pie de la letra los resultados, lo óptimo sería enlazar esta encuesta a la *beta* publicada en la *Playstore* para así llegar a más gente, y eliminar el factor cercanía con el desarrollador, y así obtener resultados mas *objetivos*.

Capítulo 10

Conclusiones y trabajo futuro

En este apartado explicaremos las conclusiones sacadas tras el desarrollo del proyecto, y se tratará también el posible trabajo futuro.

10.1 Conclusiones

Aunque no se han conseguido todos los objetivos establecidos en un principio faltando las funcionalidades relacionadas con las imágenes, por problemas de tiempos principalmente causado por la disponibilidad en cuanto a tiempos del alumno y que la curva de aprendizaje al principio del desarrollo de las iteraciones fue más compleja de lo que se esperaba, se han alcanzado una serie de objetivos con los que estar orgulloso.

Estos objetivos alcanzados podríamos dividirlos entre la conexión a servicios externos, y la propia aplicación. En cuanto a la conexión con servicios externos, los logros alcanzados han sido los siguientes:

- Se ha creado un proyecto integrado con *Firestore* montando toda la jerarquía de datos desde la aplicación. Se ha consiguido trabajar en todo momento con datos en tiempo real. Se ha prestado especial atención al diseño modular de la estructura de datos, por lo que sería muy sencillo hacer cambios y adaptaciones a otros ámbitos.

- El sistema de autenticación es rápido y sencillo de usar mediante la cuenta de Google que todo móvil Android debe tener. Además, Firebase es flexible en ese sentido y puede ser sustituido por cualquier otro sistema de autenticación si fuese necesario.

En lo relevante a la aplicación, podemos definir los logros principalmente entre objetivos funcionales, y objetivos no funcionales. Los objetivos funcionales alcanzado han sido:

- Autenticación con una cuenta de Google. Se pueden utilizar distintas cuentas, aunque sólo una en cada momento.

- Permitir a los usuarios no autenticados ver la información disponible, impidiendo que puedan crear y/o editarla.
- Visualizar a partir de la posición del dispositivo las localizaciones cercanas creadas por el usuario y también por el resto de usuarios.
- Poder crear, editar y borrar localizaciones, como también hacer lo mismo con las valoraciones, pudiendo visualizar las de otros usuarios.
- Crear una aplicación cuya información se alimenta de forma colaborativa con la interacción de los usuarios.
- Poder filtrar las localizaciones que se quieran encontrar a partir de una serie de parámetros que el usuario puede editar.

Por otra parte los objetivos no funcionales conseguidos han sido:

- Crear una interfaz intuitiva y atractiva para el usuario.
- Arquitectura MVVM, la recomendada actualmente por Google, es una arquitectura fácil de implementar que aporta muchas herramientas a la hora de trabajar con información en tiempo real.
- La arquitectura MVVM permite una adaptación muy fácil a la hora de cambiar uno de sus módulos, por lo que es muy útil si en el futuro quisiéramos cambiar de base de datos, o quisieramos aplicar la misma filosofía de la aplicación, pero para otra utilidad.
- Buen rendimiento en cuanto a tiempo de respuesta de interfaz y con servicios externos.
- Los recursos de sistema no se ven sobreexplotados.

En resumen, se ha desarrollado una aplicación para geolocalizar aseos públicos que funciona de manera colaborativa, y que se ha basado en un diseño que permite la flexibildiad tanto en cambios de tecnologías y/o contenidos. Además se ha logrado realizar una interfaz muy dinámica e intuitiva, lo que atrae a nuevos usuarios, y hace su uso muy simple.

Destacar el gran manejo y fluidez que se han ido adquiriendo a lo largo del desarrollo de los *sprints* con las distintas tecnologías y/o lengüajes de programación que en un principio el desarrolador no tenía ningún conocimiento, asunto que se ve reflejado a la perfección en cómo se ha ido desarrollando el proyecto.

Por último, hacer énfasis también en la gran flexibilidad que aportó *extremme programming* frente a las distintas vicisitudes que se fueron dando a lo largo del proyecto, principalmente en lo referente a los retrasos que se fueron presentando.

10.2 Trabajo futuro

En el momento actual del proyecto, para acabar lo que se buscaba los posibles pasos siguientes serian.

- Desarrollar el apartado de *Imágenes* que quedó sin acabar y así completar los objetivos iniciales del proyecto.
- Seguir haciendo pruebas por un tiempo con la versión *beta* publicada en la *PlayStore* e ir corrigiendo posibles bugs que puedan aparecer, y cuando se tenga una versión que se considere estable, sacarla a producción en la *Store*.

En cuanto a nuevas funcionalidades o trabajo futuro sobre la aplicación, sacadas de las encuestas de calidad a usuarios y de ideas propias, las posibles vertientes serían las siguientes.

- Añadir la posibilidad de que el usuario no conteste a algún parámetro en la valoración y no cuente para los valores finales que se muestren.
- Poder editar la ubicación de una localización.
- Poder crear *localizaciones* a partir de un punto ya existente en el mapa.
- Poder compartir localizaciones.
- Elaborar un infraestructura que permita dar *me gusta*, o alguna alternativa, a las *valoraciones* de otros usuarios, y a partir de estos valores ponderarlas para las que más *aprobación* tenga de otros usuarios, pese más a la hora de calcular los valores que se muestran de cada localización.
- Haciendo uso de la *API* de GoogleMaps, hacer una carga en la base de datos de localizaciones como *bares*, *restaurantes*, o distintos puntos de interés donde puedan encotnrarse aseos públics, de tal manera que el usuario sólo tenga que valorar estos puntos y crear otros que fuesen muy concretos y no aparecieran.
- Desarrollar y adaptar la aplicación para otras tecnologías que podrían llegar a ser útiles, como *Swift* o incluso una pequeña aplicación Web, haciendo que todo esté enlazado.
- Desarrollar una nueva app que almacene información y valoraciones de usuarios sobre algo diferente a los aseos públicos, como pueden ser sitios donde pasear a tu mascota, o pabellones deportivos, y ver cómo se adapta la infraestructura y si realmente la complejidad de adaptarlo es tan baja como se estima.

Apéndices

Apéndice A

Instalación y uso

En este apéndice, se explica brevemente cómo instalar la aplicación y se facilita un manual de uso.

A.1 Instalación

Para poder instalar la aplicación, hay dos maneras: Instalarla desde la *Playstore* o instalar el archivo *apk* incluido en el DVD.

Ahora mismo hay una versión *beta* de la aplicación bajo el nombre de *Popadvisor*, solo sería cuestión de buscarla e instalarla.

Para la segunda opción, lo único necesario sería tener descargado el *apk* en el dispositivo móvil en el que se van a hacer las pruebas, y tener habilitada la opción de *Habilitar la instalación de aplicaciones de orígen desconocido*, usualmente en el apartado de Seguridad de los ajustes, y luego ejecutar el archivo.

A.2 Manual de uso

En este apartado explicaremos cómo se utiliza la aplicación. Los apartados están ordenados de tal manera que sea sencillo seguirlos mientras se usa la aplicación por primera vez.

A.2.1 Inicio

Al iniciar la aplicación, lo primero que se nos muestra es una *Intro* con dos *slides*, las cuales podemos pasar de una a otra deslizando, y luego ir directamente a la pantalla de *login*, donde podremos decidir si iniciar sesión, o entrar a la aplicación sin hacerlo. A efectos prácticos, en este manual nos centraremos en el caso de haberse autenticado, ya que el otro caso es el mismo pero con menos funcionalidades. Un ejemplo de estas *vistas* lo podemos ver en la figura A.1.

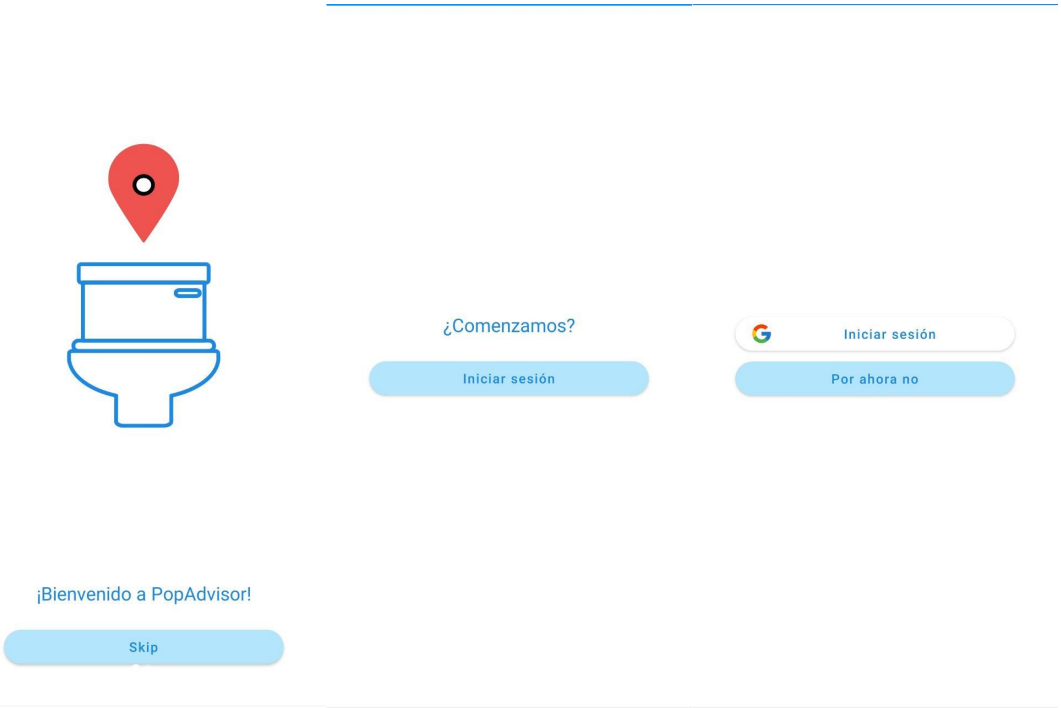


Figura A.1: Capturas pantalla de inicio.

Al pulsar en *Iniciar sesión* se abrirá un diálogo con el que seleccionaremos con qué cuenta entraremos, y al acabar nos dirigirá a la pantalla de *Home*.

A.2.2 Home

En esta parte de la aplicación, se nos muestran dos botones: *Baños cerca*, *Filtrar* y un pequeño *widget* arriba a la izquierda, el cual nos abrirá un menu lateral con la información de la sesión y la opción de hacer *logout*. Podemos ver cómo es esta pantalla en la figura A.2.

Los otros dos botones nos llevarán a otras pantallas, el primero a la vista donde se encuentra el mapa y el segundo a una serie de filtros para realizar una búsqueda.

A.2.3 Mapa

Al abrirse el mapa, vemos un menu menú inferior, las cuales la priemra opción nos dirigirá a *home*, la segunda es en la que nos encontramos, y la tercera a *filtrar*. En esta pantalla, si pulsamos en una de las localizaciones que se nos muestra, se abrirá un pequeño *layout* que nos mostrará información básica sobre la localización, si queremos ver más sobre esta localización, solo tenemos que deslizar hacia arriba. Otra de las opciones disponibles en esta pantalla es la de crear localizaciones, para ello solo tenemos que mantener pulsado en el punto deseado, y se nos abrirá la ventana para el creado. Estas tres vistas las podemos ver en la figura A.3.

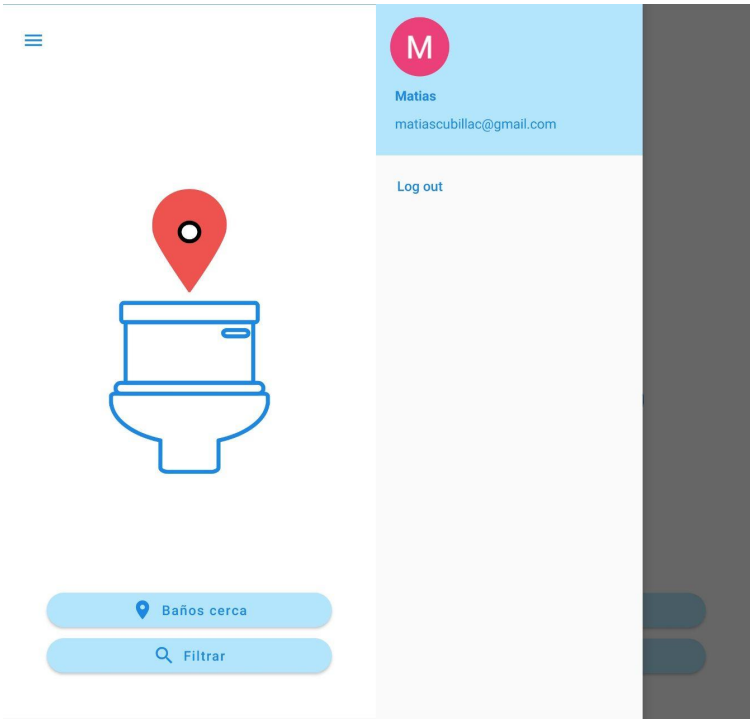


Figura A.2: Capturas de la pantalla *Home*.

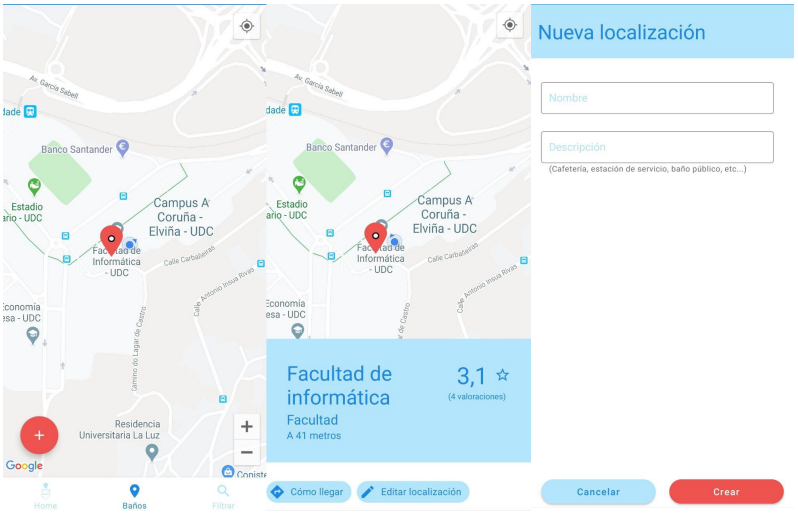


Figura A.3: Capturas de las pantallas *Mapa*.

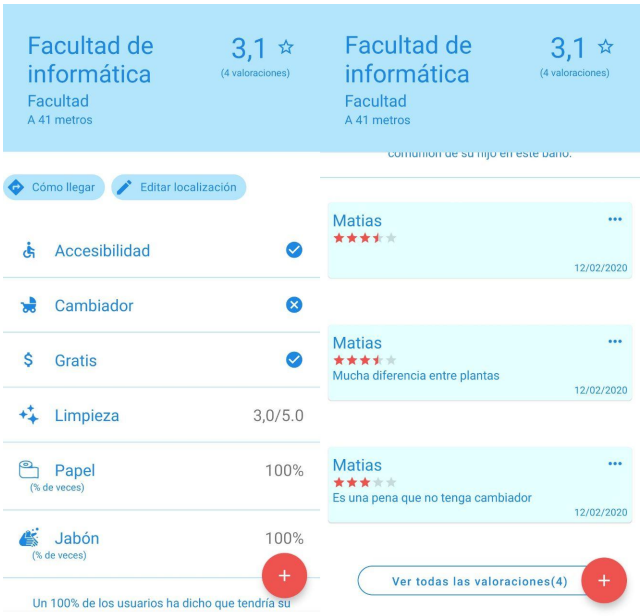


Figura A.4: Capturas de las pantallas *Localización*.

A.2.4 Localización

Al deslizar hacia arriba, se nos abrirá la información de la localización, donde primero nos encontraremos la información propiamente de la localización, y si seguimos deslizando una lista de las últimas tres valoraciones asociadas, si queremos ver el resto, solo hay que pulsar el botón que se encuentra justo debajo de la lista, que solo aparecerá en caso de que haya más de tres valoraciones. En la figura A.4 podemos ver esta pantalla.

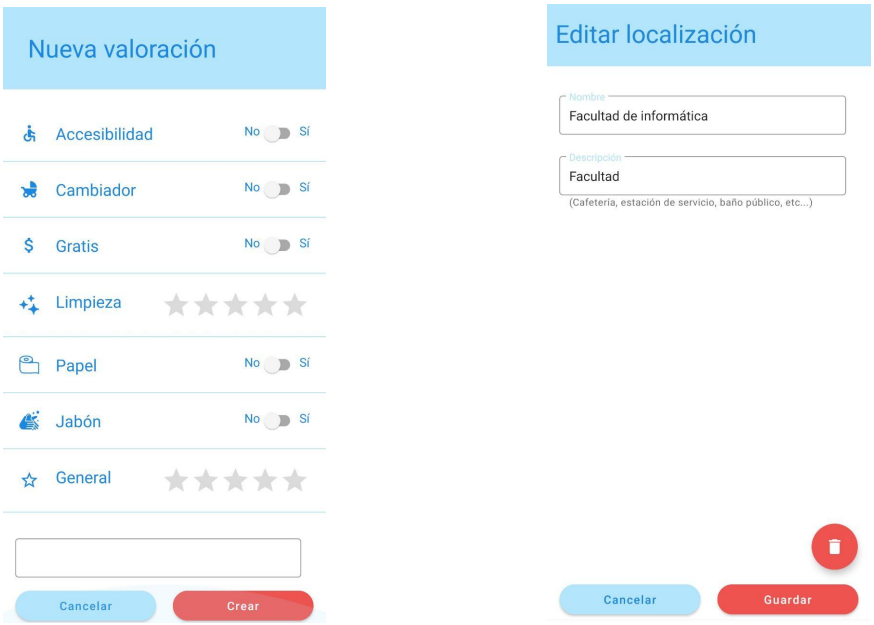
Si deseamos crear una nueva valoración, solo hay que pulsar el más rojo que se encuentra debajo a la derecha y nos abrira la pantalla de la figura A.5a.

Si queremos saber cómo llegar a esta localización hay que pulsar el botón *Cómo llegar* que se encuentra justo debajo de la información básica.

Si queremos editar la información básica, hay que pulsar el botón *Editar localización* que se encuentra junto a *Cómo llegar*, y nos abrirá la vista de la figura A.5b.

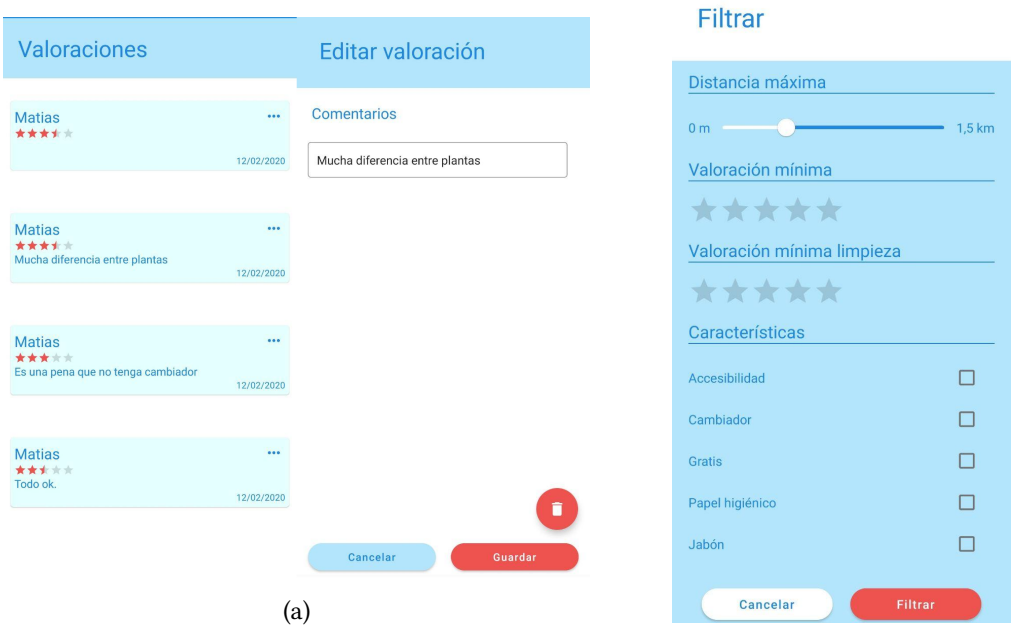
A.2.5 Valoraciones

En caso de haber pulsado el botón de *Ver todas las valoraciones*, nos abrirá una nueva ventana con una lista de todas ellas. A mayores, si queremos editar una, solo tenemos que pulsar los tres puntos de arriba a la derecha, que estarán disponbiles en la lista de todas las valoraciones como en la lista de las últimas tres de la *Localización*, lo cual nos abrirá una ventana para la edición que podemos ver en la figura A.6a junto con la lista de valoraciones que acabamos de hablar.



(a) (b)

Figura A.5: Capturas de *Nueva valoración* y *Editar localización*.



(a) (b)

Figura A.6: Capturas de las pantallas *valoraciones* y *filtrar*.

A.2.6 Filtrar

Por último, desde la pantalla de *Filtrar* podemos indicar a partir de los parámetros disponibles, qué localizaciones queremos ver, lo cual realizará una búsqueda y nos devolverá a la pantalla del mapa pero con las localizaciones que coincidan con nuestra búsqueda. En la figura A.6b podemos ver la pantalla de filtrado.

Apéndice B

Casos de uso

A continuación se detallan los casos de uso especificados en el apartado 4.4, Casos de uso

Caso de uso: Autenticarse	
Id	1
Breve descripción	El usuario se autentica en la aplicación.
Actores primarios	Usuario sin autenticar.
Precondiciones	Ninguna.
Postcondiciones	1. Las credenciales seleccionadas son válidas.

Tabla B.1: Caso de uso ”Autenticar”

Caso de uso: Cargar Localizaciones Cercanas	
Id	2
Breve descripción	El usuario carga las localizaciones que se encuentran cerca de su posición actual.
Actores primarios	Usuario sin autenticar, usuario autenticado y usuario moderador.
Precondiciones	1. El sistema tiene acceso a la posición actual del usuario.
Postcondiciones	1. Se cargan correctamente las localizaciones de la misma ciudad y país que se encuentran a menos de 500 metros de la posición del usuario.

Tabla B.2: Caso de uso ”Cargar Localizaciones Cercanas”

Caso de uso: Filtrar Localizaciones	
Id	3
Breve descripción	El usuario selecciona una serie de parámetros y a partir de ellos se hace una búsqueda.
Actores primarios	Usuario sin autenticar, usuario autenticado y usuario moderador.
Precondiciones	1. El sistema tiene acceso a la posición actual del usuario.
Postcondiciones	1. Las localizaciones se cargan correctamente.

Tabla B.3: Caso de uso ”Filtrar localizaciones”

Caso de uso: Cargar Localización	
Id	4
Breve descripción	El usuario selecciona una localización y se cargan los valores asociados a esta.
Actores primarios	Usuario sin autenticar, usuario autenticado y usuario moderador.
Precondiciones	1. Haber cargado las localizaciones cercanas o haber realizado una búsqueda.
Postcondiciones	1. La información asociada se carga correctamente

Tabla B.4: Caso de uso ”Cargar Localización”

Caso de uso: Denunciar Imagen	
Id	5
Breve descripción	El usuario podrá denunciar una imagen por contenido inapropiado o contenido erróneo.
Actores primarios	Usuario sin autenticar, usuario autenticado y usuario moderador.
Precondiciones	1. Haber cargado una localización.
Postcondiciones	1. Se hace un borrado lógico de la imagen, y se genera un aviso al administrador del sistema.

Tabla B.5: Caso de uso ”Denunciar Imagen”

Caso de uso: Cargar Valoraciones	
Id	6
Breve descripción	El usuario carga la lista de valoraciones asociadas a una localización.
Actores primarios	Usuario sin autenticar, usuario autenticado y usuario moderador.
Precondiciones	1. Haber cargado la información asociada a una localización
Postcondiciones	1. Las valoraciones se cargan correctamente

Tabla B.6: Caso de uso ”Cargar Valoraciones”

Caso de uso: Crear Localización	
Id	7
Breve descripción	El usuario selecciona una posición en el mapa, asigna un nombre y una descripción, y crea una nueva localización.
Actores primarios	Usuario autenticado y usuario moderador.
Precondiciones	1. Se han cargado las localizaciones cercanas o se ha realizado una búsqueda.
Postcondiciones	1. La localización se crea correctamente.

Tabla B.7: Caso de uso ”Crear Localización.”

Caso de uso: Cómo llegar	
Id	8
Breve descripción	Los usuarios podrán saber cómo llegar a una localización.
Actores primarios	Usuario sin autenticar, usuario autenticado y usuario moderador.
Precondiciones	1. Se ha cargado la información asociada a una localización.
Postcondiciones	1. Se indica al usuario cómo llegar a esa localización.

Tabla B.8: Caso de uso ”Cómo llegar.”

Caso de uso: Editar Localización	
Id	9
Breve descripción	Los usuarios podrán editar el nombre y/o la descripción de una localización.
Actores primarios	Usuario autenticado.
Precondiciones	1. Se ha cargada la información de una localización 2. El usuario ha creado la localización que quiere editar
Postcondiciones	1. La localización se edita correctamente

Tabla B.9: Caso de uso ”Editar Localización”

Caso de uso: Borrar Localización	
Id	10
Breve descripción	Los usuarios podrán borrar una localización.
Actores primarios	Usuario autenticado.
Precondiciones	1. Se ha cargada la información de una localización 2. El usuario ha creado la localización que quiere eliminar.
Postcondiciones	1. La localización se elimina correctamente

Tabla B.10: Caso de uso ”Borrar Localización”

Caso de uso: Subir Imagen	
Id	11
Breve descripción	Los usuarios podrán subir imágenes de las localizaciones.
Actores primarios	Usuario autenticado, usuario moderador.
Precondiciones	1. Se ha cargada la información de una localización
Postcondiciones	1. Se sube la imagen correctamente

Tabla B.11: Caso de uso ”Subir Imagen”

Caso de uso: Eliminar Imagen	
Id	12
Breve descripción	Los usuarios podrán eliminar imágenes de las localizaciones.
Actores primarios	Usuario autenticado
Precondiciones	1. Se ha cargada la información de una localización 2. El usuario ha subido la imagen que desea borrar.
Postcondiciones	1. Se borra la imagen correctamente

Tabla B.12: Caso de uso ”Borrar Imagen”

Caso de uso: Crear valoración	
Id	13
Breve descripción	Los usuarios podrán añadir valoraciones a las localizaciones, de las cuáles se calcularán los parámetros que se enseñará al cargarse las localizaciones. Las valoraciones constarán de una serie de parámetros que medirán cuestiones concretas y generales de la experiencia de los usuarios sobre esa localización.
Actores primarios	Usuario moderador.
Precondiciones	1. Se ha cargada la información de una localización
Postcondiciones	1. La valoración se crea correctamente.

Tabla B.13: Caso de uso ”Crear Valoración”

Caso de uso: Editar Valoración	
Id	14
Breve descripción	Los usuarios podrán editar los comentarios de las valoraciones.
Actores primarios	Usuario autenticado
Precondiciones	1. Se ha cargado la información de una localización o la lista de valoraciones de una localización. 2. El usuario ha creado la valoración que desea editar.
Postcondiciones	1. Se edita la valoración correctamente.

Tabla B.14: Caso de uso ”Editar Valoración”

Caso de uso: Borrar Valoración	
Id	15
Breve descripción	Los usuarios podrán borrar las valoraciones.
Actores primarios	Usuario autenticado
Precondiciones	1. Se ha cargado la información de una localización o la lista de valoraciones de una localización. 2. El usuario ha creado la valoración que desea eliminar.
Postcondiciones	1. Se elimina la valoración correctamente.

Tabla B.15: Caso de uso ”Borrar Valoración”

Caso de uso: Editar Localización Moderador	
Id	16
Breve descripción	Los usuarios moderadores podrán editar el nombre y/o la descripción de una localización.
Actores primarios	Usuario moderador.
Precondiciones	1. Se ha cargada la información de una localización
Postcondiciones	1. La localización se edita correctamente

Tabla B.16: Caso de uso ”Editar Localización Moderador”

Caso de uso: Borrar Localización Moderador	
Id	17
Breve descripción	Los usuarios moderadores podrán borrar una localización.
Actores primarios	Usuario moderador.
Precondiciones	1. Se ha cargada la información de una localización
Postcondiciones	1. La localización se elimina correctamente

Tabla B.17: Caso de uso ”Borrar Localización Moderador”

Caso de uso: Eliminar Imagen Moderador	
Id	18
Breve descripción	Los usuarios moderadores podrán eliminar imágenes de las localizaciones.
Actores primarios	Usuario moderador.
Precondiciones	1. Se ha cargada la información de una localización
Postcondiciones	1. Se borra la imagen correctamente

Tabla B.18: Caso de uso ”Borrar Imagen Moderador”

Caso de uso: Editar Valoración Moderador	
Id	19
Breve descripción	Los usuarios moderadores podrán editar los comentarios de las valoraciones.
Actores primarios	Usuario moderador.
Precondiciones	1. Se ha cargado la información de una localización o la lista de valoraciones de una localización.
Postcondiciones	1. Se edita la valoración correctamente.

Tabla B.19: Caso de uso ”Editar Valoración Moderador”

Caso de uso: Borrar Valoración Moderador	
Id	20
Breve descripción	Los usuarios moderadores podrán borrar las valoraciones.
Actores primarios	Usuario moderador.
Precondiciones	1. Se ha cargado la información de una localización o la lista de valoraciones de una localización.
Postcondiciones	1. Se elimina la valoración correctamente.

Tabla B.20: Caso de uso ”Borrar Valoración Moderador”

Apéndice C

Contenido del DVD

En el dvd se incluyen los siguientes elementos:

- El instalador de la aplicación en formato APK.
- La memoria del proyecto en formato PDF.
- Un archivo RAR que contendrá el código fuente del proyecto.
- El código fuente de la memoria en formato ZIP.

Bibliografía

[1] "Tripadvisor," Último acceso: 29 de enero de 2020. [En línea]. Disponible en: https://www.tripadvisor.es/

[2] "Google maps," Último acceso: 29 de enero de 2020. [En línea]. Disponible en: https://www.google.com/maps/about/

[3] "Foursquare," Último acceso: 29 de enero de 2020. [En línea]. Disponible en: https://play.google.com/store/apps/details?id=com.joelapenna.foursquared&hl=es

[4] "Waze," Último acceso: 29 de enero de 2020. [En línea]. Disponible en: https://www.waze.com/

[5] "Where is public toilet," Último acceso: 29 de enero de 2020. [En línea]. Disponible en: https://play.google.com/store/apps/details?id=sfcapital.publictoiletinsouthaustralia&hl=es

[6] "Aseos cerca," Último acceso: 29 de enero de 2020. [En línea]. Disponible en: https://play.google.com/store/apps/details?id=de.pnpq.toiletlocator&hl=es

[7] "Guía de arquitectura de apps," Último acceso: 29 de enero de 2020. [En línea]. Disponible en: https://developer.android.com/jetpack/docs/guide

[8] "Aspectos fundamentales de la aplicación," Último acceso: 29 de enero de 2020. [En línea]. Disponible en: https://developer.android.com/guide/components/fundamentals.html

[9] "Firebase," Último acceso: 29 de enero de 2020. [En línea]. Disponible en: https://firebase.google.com/docs?authuser=0

[10] "Mongodb," Último acceso: 03 de febrero de 2020. [En línea]. Disponible en: https://docs.mongodb.com/cloud/

[11] "Postgresql," Último acceso: 03 de febrero de 2020. [En línea]. Disponible en: https://www.postgresql.org/

[12] "Firebase authentication," Último acceso: 29 de enero de 2020. [En línea]. Disponible en: https://firebase.google.com/docs/auth/?authuser=0

[13] "Agrega acceso mediante google a tu app para android," Último acceso: 30 de enero de 2020. [En línea]. Disponible en: https://firebase.google.com/docs/auth/android/google-signin?authuser=0

[14] "Firestore," Último acceso: 29 de enero de 2020. [En línea]. Disponible en: https://firebase.google.com/docs/firestore/?authuser=0

[15] "Firebase realtime database," Último acceso: 30 de enero de 2020. [En línea]. Disponible en: https://firebase.google.com/docs/database/?authuser=0

[16] "Firebase cloud storage," Último acceso: 29 de enero de 2020. [En línea]. Disponible en: https://firebase.google.com/docs/storage/?authuser=0

[17] "Lifecycle," Último acceso: 03 de febrero de 2020. [En línea]. Disponible en: https://developer.android.com/topic/libraries/architecture/lifecycle

[18] "Kodein," Último acceso: 03 de febrero de 2020. [En línea]. Disponible en: https://kodein.org/Kodein-DI/?6.5/getting-started

[19] "Dagger," Último acceso: 03 de febrero de 2020. [En línea]. Disponible en: https://dagger.dev/

[20] "Koin," Último acceso: 03 de febrero de 2020. [En línea]. Disponible en: https://insert-koin.io/

[21] "Appintro," Último acceso: 03 de febrero de 2020. [En línea]. Disponible en: https://github.com/AppIntro/AppIntro

[22] "Material design," Último acceso: 03 de febrero de 2020. [En línea]. Disponible en: https://material.io/design/

[23] "Documentación de lottie," Último acceso: 11 de febrero de 2020. [En línea]. Disponible en: https://airbnb.io/lottie/#/

[24] "Kotlin," Último acceso: 03 de febrero de 2020. [En línea]. Disponible en: https://kotlinlang.org/

[25] "Xml," Último acceso: 03 de febrero de 2020. [En línea]. Disponible en: https://developer.mozilla.org/es/docs/Web/XML/Introducción

[26] "Android studio," Último acceso: 03 de febrero de 2020. [En línea]. Disponible en: https://developer.android.com/studio/intro

[27] "Trello," Último acceso: 03 de febrero de 2020. [En línea]. Disponible en: https://trello.com/

[28] "Figma," Último acceso: 03 de febrero de 2020. [En línea]. Disponible en: https://www.figma.com/

[29] "Inkscape," Último acceso: 03 de febrero de 2020. [En línea]. Disponible en: https://inkscape.org/es/

[30] "Documentación de aftereffects," Último acceso: 11 de febrero de 2020. [En línea]. Disponible en: https://helpx.adobe.com/es/support/after-effects.html?promoid=RBS7NR3C&mv=other

[31] "Xml," Último acceso: 03 de febrero de 2020. [En línea]. Disponible en: https://es.overleaf.com/learn

[32] "Magicdraw," Último acceso: 03 de febrero de 2020. [En línea]. Disponible en: https://docs.nomagic.com/display/MD185/MagicDraw+Documentation

[33] "Dia diagrams," Último acceso: 03 de febrero de 2020. [En línea]. Disponible en: http://dia-installer.de/doc/index.html.en

[34] C. De Parga, *UML. Aplicaciones en Java y C++*. Grupo Editorial RA-MA, 2015.

[35] L. Smith, *Agile Software Development with C#, Scrum, Extreme Programming, and Kanban Revised Edition*. CreateSpace Independent Publishing Platform, 2019.

[36] F. Charte and F. Ojeda, *SQL*, ser. Guias practica / Practical Guide. ANAYA, 2009.

[37] "Consultas de agreación en firestore," Último acceso: 08 de febrero de 2020. [En línea]. Disponible en: https://firebase.google.com/docs/firestore/solutions/aggregation

[38] "Firebase pricing," Último acceso: 05 de febrero de 2020. [En línea]. Disponible en: https://firebase.google.com/pricing?authuser=0

[39] "Guía para integrar sdk de crashlytics," Último acceso: 10 de febrero de 2020. [En línea]. Disponible en: https://firebase.google.com/docs/crashlytics/get-started?authuser=0&platform=android

[40] "Guía para integrar sdk de performance," Último acceso: 10 de febrero de 2020. [En línea]. Disponible en: https://firebase.google.com/docs/perf-mon/get-started-android?hl=es-419

[41] "Firebase testlab," Último acceso: 10 de febrero de 2020. [En línea]. Disponible en: https://firebase.google.com/docs/test-lab?hl=es